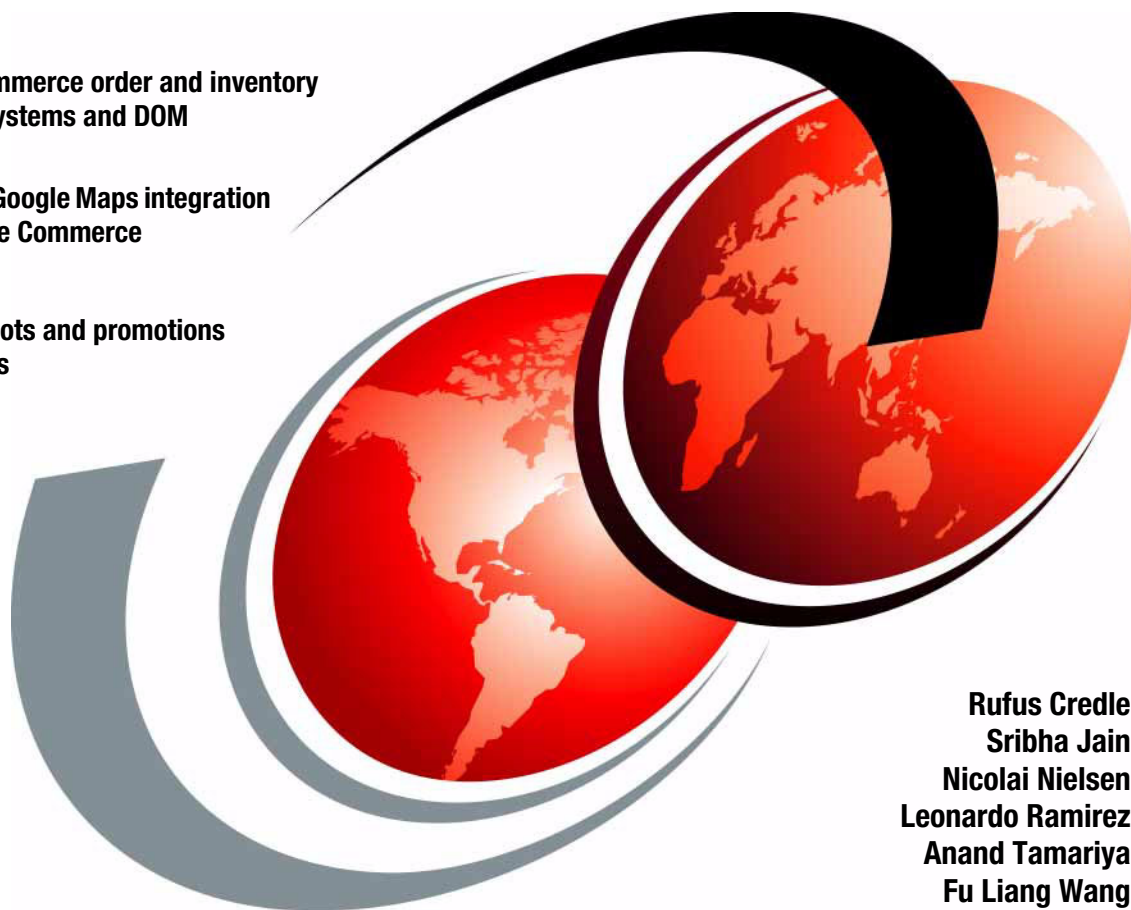


Building Multichannel Applications with WebSphere Commerce

WebSphere Commerce order and inventory management systems and DOM

MapQuest and Google Maps integration with WebSphere Commerce

e-Marketing Spots and promotions for mobile users



Rufus Credle
Sribha Jain
Nicolai Nielsen
Leonardo Ramirez
Anand Tamariya
Fu Liang Wang



International Technical Support Organization

**Building Multichannel Applications with
WebSphere Commerce**

February 2010

Note: Before using this information and the product it supports, read the information in “Notices” on page ix.

First Edition (February 2010)

This edition applies to IBM WebSphere Commerce V7, IBM WebSphere Application Server V7, IBM WebSphere Commerce V6, and IBM WebSphere Commerce V7 Developer Edition.

© Copyright International Business Machines Corporation 2010. All rights reserved.

Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Notices	ix
Trademarks	x
 Preface	xi
The team who wrote this book	xii
Become a published author	xiv
Comments welcome	xv
 Chapter 1. Multichannel solution overview	1
1.1 Advantages of using a cross-channel solution	2
1.2 Cross-channel features	5
1.2.1 Cross-channel evolution	8
1.2.2 The most popular cross channel shopping processes	9
1.3 The IBM solution	10
1.3.1 IBM Retail Integration Framework	11
1.3.2 WebSphere Commerce integration capabilities	11
1.3.3 SOA for cross-channel integration	15
1.4 Cross-channel configurations	16
1.4.1 WebSphere Commerce and DOM	18
1.4.2 WebSphere Commerce and BPM	22
1.4.3 WebSphere Commerce on store environments	24
1.5 Summary	28
 Chapter 2. IBM WebSphere Commerce V7 features	29
2.1 Enhanced Madisons Starter Store	30
2.1.1 Madisons Starter Store	30
2.1.2 Enhanced business-to-business starter store	46
2.1.3 Madisons Mobile Starter Store add-on store archive	47
2.2 Social Commerce	52
2.2.1 Sharing on social networks	53
2.2.2 Ratings and reviews	54
2.2.3 Product and category blogs	56
2.2.4 Photo and video galleries	56
2.2.5 Social profile	57
2.2.6 Social Commerce integration	58
2.3 Marketing improvements	59
2.3.1 New promotion types	60
2.3.2 Deprecated promotion adjustments	61
2.3.3 New promotion features	61

2.3.4 Precision marketing	65
2.4 Management Center enhancements	69
2.4.1 Activity templates	70
2.4.2 Customer segments	71
2.4.3 Statistics	71
2.5 Utilities	74
2.6 Stack update	75
Chapter 3. Distributed order management integration	77
3.1 Overview of WebSphere Commerce order management and inventory management	78
3.1.1 WebSphere Commerce order management	78
3.1.2 WebSphere Commerce inventory management	81
3.2 Integration of WebSphere Commerce with the DOM solution	84
3.2.1 WebSphere Commerce DOM solution overview	84
3.2.2 Shopping flow with DOM	87
3.2.3 DOM integration structure	88
3.2.4 DOM integration outbound messages	91
3.2.5 DOM integration inbound message	95
3.2.6 Integration steps	98
3.3 Implementation of WebSphere Message Broker mediation module for DOM integration	98
3.3.1 Installation	99
3.3.2 Post-installation tasks	99
3.3.3 Create default configuration for WebSphere Message Broker	100
3.3.4 Creating mediation module in WebSphere Message Broker	102
3.3.5 Deploying mediation module	122
3.4 Implementing WebSphere Enterprise Service Bus mediation module for DOM integration	126
3.5 Configuring the DOM integration feature	126
Chapter 4. Web 2.0 storefront	129
4.1 WebSphere Commerce Web 2.0 store overview	130
4.1.1 Web 2.0 in WebSphere Commerce	130
4.1.2 Madisons Starter Store	132
4.2 Buy online, pick up in store	134
4.2.1 BOPIS overview	135
4.2.2 BOPIS in Madisons Starter Store	135
4.3 The Store Locator feature	150
4.4 MapQuest integration for Store Locator	151
4.5 MapQuest Geocoding	166
4.5.1 Install MapQuest JavaScript API proxy	166
4.5.2 Integration with IBM WebSphere Commerce	171

Chapter 5. Mobile commerce features in WebSphere Commerce V7 . . .	177
5.1 Madisons Mobile Starter Store	178
5.1.1 Mobile commerce overview in WebSphere Commerce	178
5.1.2 Madisons Mobile Starter Store	179
5.1.3 Mobile marketing.	191
5.2 Promotions	202
5.3 Integration with a map service provider for the mobile Store Locator . . .	208
5.3.1 Example: Google Map integration for iPhone	209
5.4 Buy on mobile device and select shipping address.	221
5.4.1 Standard checkout page flow for Madisons Mobile Starter Store . .	221
5.4.2 Checkout with shipping for Madisons Mobile Starter Store.	224
5.4.3 Analyze the existing Madisons Mobile Starter Store code	228
5.4.4 Design new shopping flow	239
5.4.5 Create new pages.	241
5.4.6 Modify existing pages	291
5.5 Product ratings and review	313
5.5.1 Ratings and reviews on the mobile device	313
5.5.2 Integration approach	319
5.5.3 Prepare the workspace	322
5.5.4 Create the ratings and reviews data beans.	324
5.5.5 Create the generic star display JSP	336
5.5.6 Create the JSP fragment for the Write Review button	341
5.5.7 Create the JSP fragment for the review header	343
5.5.8 Modify the product display pages	345
5.5.9 Modify the product compare page.	353
5.5.10 Create the JSP for the review list and overview	355
5.5.11 Create a page for review details	363
5.5.12 Create the JSP for writing the review	369
5.5.13 Create the command to post the review	375
5.5.14 Modify infrastructure resources.	384
5.6 Analytics	399
5.6.1 Coremetrics Web Analytics for WebSphere Commerce	399
5.6.2 Implementing mobile analytics	404
Chapter 6. Example client cross-channel solution	421
6.1 Sample client solution	422
6.1.1 Challenges for today's global retailer	424
6.1.2 Business requirements	424
6.1.3 IT requirements.	425
6.1.4 Analyzing the requirements and managing expectations	425
6.1.5 Phase 1: Aligning the business and IT processes using SOA	426
6.1.6 Phase 2: Building a cross-channel strategy	436
6.1.7 Sample scenarios using the architecture	438

6.1.8 Conclusion	441
6.2 Targeted marketing and promotion for Web and mobile channel users .	441
6.2.1 Case scenario goals	441
6.2.2 Easy Hogar y Construcción registration sample	442
6.2.3 Easy Hogar y Construcción “do it yourself” project sample	444
6.2.4 Easy Hogar y Construcción family connections strategy sample . .	448
6.3 Cross-channel precision marketing	450
6.3.1 Marketing strategy.	450
6.3.2 Indicators.	450
6.3.3 Using WebSphere Commerce features	451
6.4 Summary	453
Appendix A. Samples of WebSphere Commerce SOA Service Request and Response	455
A.1 ProcessInventoryRequirement with action code ReserveInventory request	456
A.2 ProcessInventoryRequirement with action code ReserveInventory response	460
A.3 ProcessInventory with action code CancelInventoryReservation request	464
A.4 ProcessInventoryRequirement with action code CancelInventoryReservation response.	467
A.5 ProcessOrder with action code TransferOrder request	468
A.6 ProcessOrder with action code TransferOrder response	474
A.7 SyncOrder request	474
A.8 SyncOrder response	475
A.9 ProcessInventoryReqmt_UpdateInventory ReservationsReq.xml.	477
A.10 UpdateInventoryReservationsResp_AcknowledgeInventoryReqmt.xml	480
Appendix B. Sample code	483
B.1 ReviewsDataBean sample code	484
B.2 ReviewsDetailBean sample code	491
B.3 ReviewsDataBean sample code	494
B.4 PostReview sample code	501
Appendix C. Additional material	507
Locating the Web material	507
Using the Web material	508
How to use the Web material	508
Related publications	509
IBM Redbooks publications	509

Online resources	509
How to get Redbooks.....	511
Help from IBM	511
Index	513

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.


COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at <http://www.ibm.com/legal/copytrade.shtml>

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

BladeCenter®	MQSeries®	System x®
DB2 Universal Database™	OMEGAMON®	Tivoli Enterprise Console®
DB2®	Rational®	Tivoli®
Global Business Services®	Redbooks®	WebSphere®
IBM®	Redpapers™	
Lotus®	Redbooks (logo)  ®	

The following terms are trademarks of other companies:

Oracle, JD Edwards, PeopleSoft, Siebel, and TopLink are registered trademarks of Oracle Corporation and/or its affiliates.

SAP, and SAP logos are trademarks or registered trademarks of SAP AG in Germany and in several other countries.

Java, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

Preface

This IBM® Redbooks® publication discusses the value proposition of cross-channel solutions and describes the IBM Retail Integration Framework Commerce Product Strategy solution and service-oriented architecture (SOA) as an enabler. In depth, this book describes cross-channel processes and cross-channel features and proposes scenarios and configurations to meet the challenges in a competitive environment.

This book describes the latest features and techniques of IBM WebSphere® Commerce Version 7. In it, we present an overview of the WebSphere Commerce order and inventory management systems, the distributed order management (referred to as *DOM* throughout this book) integration framework, and a sample DOM integration scenario.

We discuss the Madisons starter store (Web 2.0 storefront) and present a hands-on experience that integrates MapQuest with the WebSphere Commerce V7 Store Locator feature. We discuss how a merchant can use the mobile features that are included in WebSphere Commerce V7 to define e-Marketing Spots and promotion for mobile users. In addition, we demonstrate how to use Google Maps with the Store Locator feature on a mobile device.

We include in this book an example about how to apply WebSphere Commerce features on a cross-channel solution as applied at the Easy Hogar y Construcción home improvement retail company in South America. The scenario explains how to scale from an SOA store to a cross-channel business model.

This book is designed for use by WebSphere Commerce developers, practitioners, and solution architects in various industries.

The team who wrote this book

This book was produced by a team of specialists from around the world working at the International Technical Support Organization (ITSO), Raleigh Center.

Rufus Credle is a Certified Consulting IT Specialist at the ITSO, Raleigh Center. In his role as Project Leader, he conducts residencies and develops IBM Redbooks and Redpapers™ publications about network operating systems, enterprise resource planning (ERP) solutions, voice technology, high availability, clustering solutions, Web application servers, pervasive computing, IBM and OEM e-business applications, WebSphere Commerce, IBM industry technology, System x®, and IBM BladeCenter®. Rufus' various positions during his IBM career include assignments in administration and asset management, systems engineering, sales and marketing, and IT services. He has a BS degree in business management from Saint Augustine's College. Rufus has been employed at IBM for 29 years.

Sribha Jain is an IT Specialist with IBM Global Business Services®, India. She has 6 years of experience in WebSphere Commerce Server. Her areas of expertise include WebSphere Commerce, Web 2.0 solutions, and Integration technologies. She holds a degree in Masters of Computer Management from Pune University.

Nicolai Nielsen is a Senior IT Specialist at IBM Hursley Software Lab, United Kingdom. He has been with the WebSphere Commerce Technology Practice of IBM Software Services for WebSphere since 2006. Before that, he worked as a WebSphere Commerce consultant with IBM Global Services, Denmark. Nicolai holds a M.Sc in Engineering from the Technical University of Denmark and has more than 8 years experience in designing and implementing WebSphere Commerce based e-commerce solutions. He has co-authored several IBM Redbooks publications on the subjects of WebSphere Commerce and application development.

Leonardo Ramirez is Regional Chief Architect for Easy Hogar y Construcción in South America. Prior to this, he was the Enterprise Architect for Easy Hogar y Construcción and led the architecture of the first SOA-enabled store in Latin America using IBM Retail Integration Framework and the ARTS SOA Blueprint and Best Practices. Prior to joining Easy Hogar y Construcción, he worked with several IBM Business Partners in Latin America. He is the recipient of several awards for innovation, and he has 11 years of experience working as an e-business solution advisor and as an Enterprise and an SOA Architect.

Anand Tamariya is an Application Architect with IBM Global Business Services. He specializes in consulting on and implementing cross-channel retail solutions using IBM software stack. His expertise includes WebSphere Commerce, Web

2.0 solutions, SOA, and integration technologies. He has a Bachelor of Technology degree from Indian Institute of Technology (IIT) Kharagpur, India.

Fu Liang Wang is a WebSphere Commerce Developer in the IBM China Software Development Lab, Beijing. He has 4 years of experience in e-Commerce field. His areas of expertise include Java™ EE, SOA, Web 2.0, and Dojo. He holds a master degree in Computer Science from Chinese Academy of Sciences.



Figure 1 Left to right: Rufus Credle, Leonardo Ramirez, Anand Tamariya, Nicolai Neilsen, Sribha Jain, and Fu Liang Wang

Thanks to the following people for their contributions to this project:

Tamikia Barrow, Margaret Ticknor, Jim Hoy
International Technical Support Organization, Raleigh Center

Michael Au, Manager, WebSphere Commerce Foundation Development
IBM Toronto

Alex Shum, Software Developer, WebSphere Commerce Solutions Development
IBM Toronto

John McLean, Development Manager, WebSphere Commerce Promotions,
Marketing and Analytics
IBM Toronto

Emir Garza, Consulting IT Specialist; Master Inventor
IBM UK

Dick Hrabik, IBM AIM Customer Programs, Program Manager WebSphere
Commerce
IBM Austin

Madhu Chetuparambil (Architects from sMASH team), STSM, WebSphere
Commerce
IBM Pittsburgh

Marco Deluca, Business Solution Architect, WebSphere Commerce Services
IBM Toronto

N Krishnan, WebSphere Services
IBM India

Kirk Wendland, WebSphere Commerce Specialist
IBM Toronto

Derek Norridge, WebSphere Services Engagements
IBM Toronto

EASY IT Team
EASY Colombia, Chile, Argentina

Now you can become a published author, too!

Here's an opportunity to spotlight your skills, grow your career, and become a published author—all at the same time! Join an ITSO residency project and help write a book in your area of expertise, while honing your experience using leading-edge technologies. Your efforts will help to increase product acceptance and customer satisfaction, as you expand your network of technical contacts and relationships. Residencies run from two to six weeks in length, and you can participate either in person or as a remote resident working from your home base.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our books to be as helpful as possible. Send us your comments about this book or other IBM Redbooks publications in one of the following ways:

- Use the online **Contact us** review Redbooks form found at:

ibm.com/redbooks

- Send your comments in an e-mail to:

redbooks@us.ibm.com

- Mail your comments to:

IBM Corporation, International Technical Support Organization
Dept. HYTD Mail Station P099
2455 South Road
Poughkeepsie, NY 12601-5400



Multichannel solution overview

This chapter discusses the value proposition of cross channel solutions. It describes the IBM Retail Integration Framework, Commerce Product Strategy solution and service-oriented architecture (SOA) as an enabler, as well as cross channel processes and cross channel features. It includes scenarios and configurations to fit this solution into a competitive environment.

This chapter contains the following sections:

- ▶ Advantages of using a cross-channel solution
- ▶ Cross-channel features
- ▶ The IBM solution
- ▶ Cross-channel configurations
- ▶ Summary

1.1 Advantages of using a cross-channel solution

In today's challenging environment, businesses must deal with new and emerging considerations:

- ▶ Macroeconomic conditions, such as financial crisis, growth market slow down, environmental concerns, product tracking, traceability for safety, fluctuating oil, and commodity prices.
- ▶ Smarter consumers, where consumers are more informed and have more tools and channels that help them to make smarter decisions and to play an active role in social commerce.
- ▶ Technology that includes an increasing number of mobile devices and the adoption of pervasive connectivity.
- ▶ Budgets for companies who are looking for smart investments and who are developing strategic plans, because every channel has its own budget, strategy, and infrastructure.
- ▶ Consistency, where companies have previously ignored the consistency and accuracy of product and service information across all channels throughout the consumers life cycle.

These conditions call for a cross channel solution that promotes IT business alignment to build a more efficient and flexible business infrastructure. A *cross channel solution* builds an easy interaction between a customer and the company. This interaction lets the customer choose appropriate transactions. The cross-channel solution provides a consistent message at every transaction point (Web site, kiosk, call centers, store, and mobile devices) and presents the company's promotions, processes, and policies coherently, wherever the customer might do business with the company.

In addition, with a cross-channel solution, companies can integrate throughout all touchpoints and sales channels, through back-end as well as existing systems, and beyond the four walls of the brick-and-mortar company to suppliers, fulfillment partners, customers, and the entire value chain.

The terms *multichannel solutions* and *cross channel solutions* are often used interchangeably, but there is a conceptual difference between these two.

In a cross channel solution, retailers have various channels (stores, Web sites, catalogs, call centers, kiosk, and so forth) so that customers can make purchases using a method that is convenient for them. With cross channel solutions, retailers aim for a seamless customer experience, a unified customer-centric experience that leads a whole vision throughout all channels, including mobile commerce.

So by taking advantage of multichannel solutions into a cross channel experience, a single customer interaction platform can be put in place to provide a more consistent, relevant experience for customers. Cross-channel solutions provide the following advantages over multichannel solutions:

- ▶ A single view of customer data and metrics throughout channels that aids in providing consistent data to customers (for example, credits) and better analysis of customer behavior.
- ▶ Better targeted merchandising and marketing as a result of consistent and integrated customer behavior data.
- ▶ A single inventory pool that takes care of lead times, demand-supply imbalances, and reverse logistics across all operations (buybacks, return to vendor, customer returns, and so forth).

A cross channel solution provides the following general features, which are imperative to today's retail industry:

- ▶ A seamless shopping experience to shoppers

Recent studies of consumer shopping behavior indicate that multichannel shoppers show a significantly higher value and frequency of purchase than single-channel shoppers. With a cross channel solution in place, the customer can choose how to interact and shop. A customer can browse and initiate the transaction at any point and either fulfill it in the store or ship it home seamlessly. Customers are provided with timely product availability and allocation for reliable order delivery and end-to-end visibility.

Figure 1-1 gives an high-level view of important cross channel solutions.

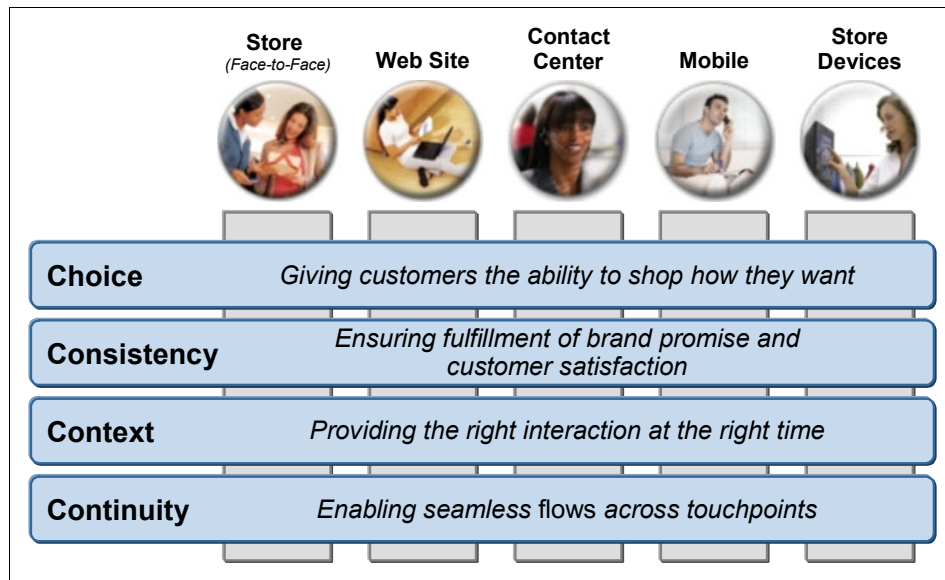


Figure 1-1 Cross-channel solutions

- ▶ A competitive edge for retailers
 - Multichannel allocation and execution functionality ensures the sale either in store or online. An option of initiating an order online and fulfilling it in store results in additional sales in most of the cases.
 - Precision multichannel marketing capabilities enable better analysis of customer attributes and buying patterns. These capabilities also enhance cross- or up-selling as well as targeted promotions throughout channels, which can increase both conversion rate and average order value by providing the following information:
 - The analytical capabilities and integrated information system of multichannel systems improve the ability to track behaviors and purchase history of customers and to use this data for targeted merchandising and marketing purposes.
 - Based on the behavioral track, the experience for customers can be personalized. In addition, you can more effectively target customers in their preferred medium with the most applicable offers and take advantage of cross-sell or up-sell opportunities.

- Consumers expect access to product information and transactional capability to be efficient, accurate, and delivered in the same way as by the retailer. Fundamentally, multichannel retailing, when delivered well, draws customers' loyalty to the brand.

When companies consider a transition from customer acquisition to customer retention, the primary focus moves from mass marketing to enabling an optimized customer experience across the multiple channels in which the consumer interacts with the company. Consumers expect a personalized experience to be available, regardless of how they interact with a business.

In this environment, interactions across channels are critical in maintaining customer retention and loyalty. For example, ordering online and picking up at the store has gone from a nice-to-have option to an expected one. Consumers expect to shop for any item, in any channel, and complete the sale seamlessly—including associated services—in any other channel.

1.2 Cross-channel features

The importance of cross channel solutions lies in providing a customer with a seamless shopping experience, thus increasing customer loyalty to the brand and reducing total operating costs by integrating previous silo systems. A complete cross channel solution takes care of all aspects of both Web and in-store shopping. Starting from pre-store tasks to post-transaction activities in store and demand generation to service and support in Web-channel, everything is covered with one integrated solution.

Figure 1-2 summarizes cross channel features for a Web-to-store solution.

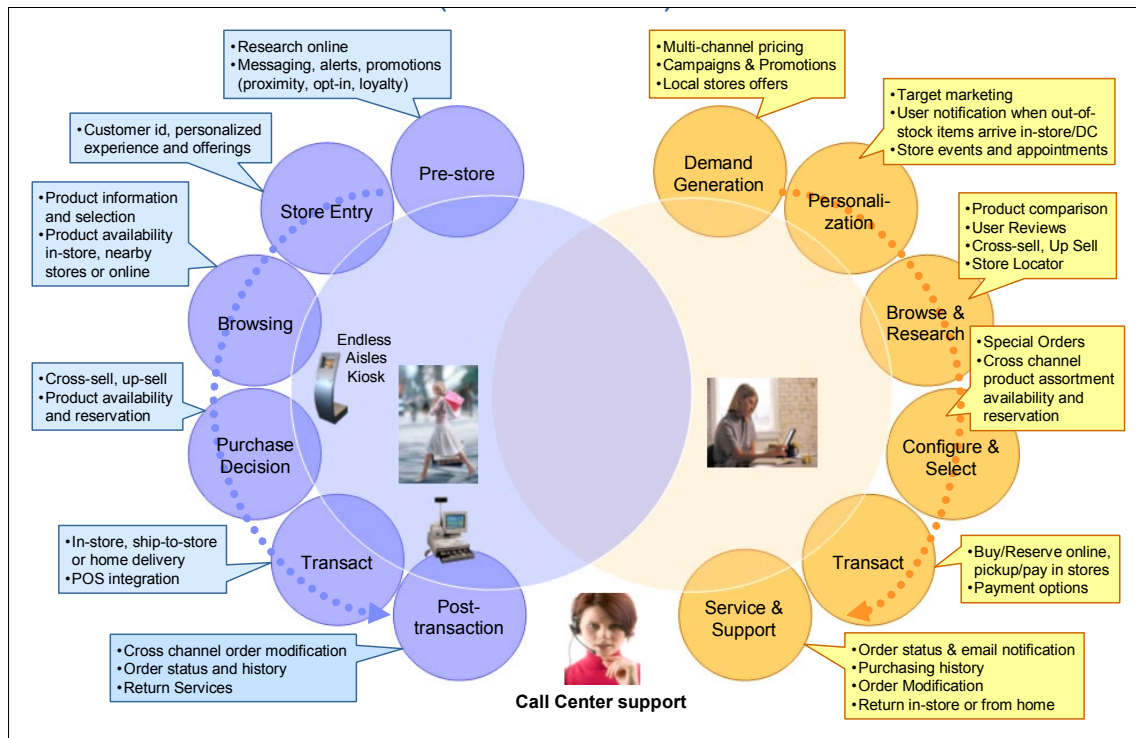


Figure 1-2 Cross-channel features (Web-to-store)

Figure 1-2 includes the following cross-channel features:

- **Pre-store activities and demand generation**
Integrated analytics, social commerce, and an inventory system help to gather information regarding supply and demand and help to review each individual product. Based on this data, the retailer can offer various campaigns, promotions, and in-store offers to increase demand.
- **Personalization**
The information regarding offers and promotions can be delivered to the customer through various points, including printed or Short Message Service (SMS) promotions as well as e-mail or in-store campaigning, depending on customer preferences. In addition, customers can be notified about out-of-stock items when those items arrive in store.

► Browsing

Customers can browse the catalog through the Web or kiosks and get the same results as in store. They can be offered personalized cross- or up-sell opportunities and promotions. They can choose to browse and transact online and can pick up merchandise in the store of their choice.

► Purchase decision

Customers can make a purchase decision based on availability, price, and cross- or up-sell opportunities. Availability is aided through a centralized inventory system and gives real-time inventory information. Pricing is also consistent throughout all channels, because that information also comes from a central system. A retailer might offer different prices for different channels, based on the demand and customer behavior.

► Transact

Various delivery and payment options are made available in a cross channel solution. A customer can buy or reserve merchandise online and then can pick up or pay in store. Customers can also choose to pay online and have the order delivered either to the store or to home. While shopping in store, a customer can choose to have the order delivered to home. They can choose to pay online or in store using various payment methods.

► Post-transaction (service and support)

Order status is available online and in store. The customer can be informed through SMS or e-mail, as per that customer's preference. The customer can choose to modify an order online or in store, which is done seamlessly for them. Products bought online can be returned in store or from home, thus providing complete comfort and flexibility to the customer. In addition, the customer's purchase history is always accessible online or in store for reference.

► Call center support

All customer and order information is available at the call center. Thus, customers can contact the call center at any time to obtain information about order or return status.

1.2.1 Cross-channel evolution

Cross-channel evolution has occurred as a natural evolution of IT technologies and standards. Table 1-1 provides a quick review of the phases that the industry has been taken to transition to a cross channel strategy.

Table 1-1 Cross-channel evolution

Phases	Description
<p>Silo systems include channels for specific transactions, such as the following transactions:</p> <ul style="list-style-type: none">► Buy online and ship to home► Buy off the shelf items in store► Call a customer service representative (CSR) to purchase	<p>This phase includes the following main pains:</p> <ul style="list-style-type: none">► Retailers operate in siloed channels, resulting in conflicting metrics and goals, thus discouraging customers. For example, the in-store sales staff does not get credit for sales on the Web.► Processes and data are duplicated across channels and are inconsistent, which might lead to unhappy customers. Thus, master data (customer, product, vendor, and so forth) that supports and streamlines retailers' operations and functionalities across channels is needed.► The siloed merchandising processes handcuff the company and contribute to long lead times, budget overruns, and regular supply and demand imbalances.
<p>Cooperation across channels provides communications and interfaces between applications such as the following applications:</p> <ul style="list-style-type: none">► Buy online and ship to store► Lookup store location	<p>With this phase, transactions are coordinated between Web, store, and call center.</p> <p>There are many point-to-point connections, or there might be Enterprise Application Integration (EAI) middleware that transforms messages using applications.</p> <p>On this phase, there is still a big impact to manage business needs, but there is little support for mobile deployment.</p>

Phases	Description
<p>Cross-channel retailing ties with previously siloed systems with an SOA-based approach.</p> <p>Companies use industry-standard reference models and enterprise architecture frameworks.</p> <p>This phase provides seamless integration for the customers where the company offers the following capabilities:</p> <ul style="list-style-type: none"> ▶ Buy online and pick up in store ▶ Reserve online and pay or pick up in store ▶ View store promotions and events online ▶ Offer online and redeem in store ▶ Buy online and return in store ▶ Buy out-of-stock items in store and ship to home 	<p>By taking advantage of multichannel solutions, a company can put in place a single customer interaction platform that provides a more consistent and more relevant experience for customers. This solution provides the following advantages:</p> <ul style="list-style-type: none"> ▶ Single view of customer data and metrics across channels, which aids in providing consistent data to customers (for example, credits) and provides better analysis of customer behavior. ▶ Better targeted merchandising and marketing as a result of consistent and integrated customer behavior data. ▶ Single inventory pool takes care of lead times, demand-supply imbalances, and reverse logistics across all operations (buybacks, return to vendor, customer returns, and so forth). ▶ Use of mobile devices to integrate the company to customer's world.

1.2.2 The most popular cross channel shopping processes

With multichannel integrated shopping solutions, retailers can focus on the integrated consumer retail experience. Retailers can differentiate their brand from competitors by taking advantage of a flexible and integrated process-based platform across marketing, selling, order capture, and fulfillment. This solution simplifies and transforms the shopping experience by making it easier for consumers to transact business across channels seamlessly.

When looking for a logical model to work with cross channel solutions, IBM defines the following top cross channel processes:

- ▶ Buy or reserve online, pick up or pay in store

This process is the key process to bridge the online and in store shopping experience. You can achieve this process in multiple way (for example, point-to-point, messaging, or process) using an SOA-based approach. When developing an enterprise architecture, you can make a step-by-step plan that lets your company adopt a roadmap for integration that is aligned with this business process. You also can use new *Commerce* functions without modification, such as store locator, local inventory, and payments.

- ▶ Return or exchange items bought online to the store

This process is an essential process for providing a complete cross channel solution. With this process, the IT business must be aligned, because there is an entire view of the consumer life cycle and IBM WebSphere Commerce V7 that is built on returning or exchanging items bought online in the store. This feature increases customer return and loyalty with a good shopping experience.

- ▶ Buy merchandise in store when an item is out-of-stock

This process is about saving the sale when items are not available. On-demand companies need to use the right tools to offer the best shopping experience for consumers. Your cross channel solution must provide a full view into the entire inventory, throughout the enterprise and other stores, to let the IT infrastructure be the framework for composite applications that do the supporting work.

- ▶ Offer and purchase additional items in store (targeted marketing)

This process helps you to understand the customer and, thus, to increase the size of the sale as well as build brand loyalty. This process is an important feature in WebSphere Commerce, because it uses granular customer information, including channel events, to purchase items in a nearby store. The customer can receive targeted marketing through SMS and social commerce.

1.3 The IBM solution

Companies ship products world wide. This global distribution process involves manufacturers, suppliers, and distributors. Thus, there is a real need to seamlessly integrate information through both internal and external networks. IBM provides a supporting infrastructure and best practices, based on industry assets, that you can use to build a robust platform that manages an efficient integration and a useful cross channel solution. IBM offers a wide breadth of products to satisfy the on-demand business requirements:

- ▶ Application infrastructure
- ▶ Application and process integration
- ▶ Portal, commerce
- ▶ Collaboration
- ▶ Messaging
- ▶ Database integration
- ▶ Enterprise security
- ▶ Pervasive computing
- ▶ End-to-end application development

WebSphere Commerce takes advantage of and integrates with these middleware products and technologies to deliver and amplify their value within an e-commerce and multichannel context.

1.3.1 IBM Retail Integration Framework

IBM Retail Integration Framework is a structure that delivers integration focused design and software assets to help make complex integration simpler and faster to retailers. By adopting the IBM Retail Integration Framework strategy, you can take advantage of a enterprise architecture and an SOA-based approach, which enable you to use business process management methods to view the IT environment as linked, repeatable business tasks or services. With IBM Retail Integration Framework, you can make your IT environment more responsive to the needs of the business.

IBM Retail Integration Framework is a growing set of pre-built application integration assets that address the following types of high-value retail domains:

- ▶ Managing master data for merchandising and supply chain: A platform to maintain uniform and consistent master data of your main elements, such as products, locations, and vendors.
- ▶ Cross-channel selling: Enable a cross channel retailing solution that allows you to expose retail capabilities to your customers across multiple touchpoints.
- ▶ Supply chain visibility: Take advantage of seamless visibility into supply chain events across channels to manage those events and to help reduce operating costs.
- ▶ Managing master data for customers: Implement a platform that helps you maintain uniform and consistent master data regarding customers.
- ▶ Integrated retail analytics: Inject customer insights into the retail processes through integrated retail analytics, which helps to make insights actionable.
- ▶ Total store solutions: Integrate and improve management in the store environment while accelerating the introduction of innovative solution.

1.3.2 WebSphere Commerce integration capabilities

WebSphere Commerce lets a company deliver a continuous and personalized customer experience, regardless of the channel in which a customer participates. It lets the company manage different scenarios such as business-to-business, business-to-consumer, and other specialized products to integrate and improve the line of business. Every aspect of WebSphere Commerce is designed to take advantage of the appropriate components of the IBM middleware platform.

Rather than developing its own solutions for the services and capabilities that are needed to support a cross channel commerce environment (such as database, portal, integration, security, and so forth), WebSphere Commerce takes advantage of the leading products within the IBM middleware portfolio to provide those capabilities. WebSphere Commerce is pre-integrated with these middleware products and includes the appropriate components, complete with integrated installation, configuration, and operations.

In addition, WebSphere Commerce provides accelerators that make these components relevant and useful within a commerce environment. For example, WebSphere Commerce includes pre-integrations, reference applications, adaptors, and tools to accelerate integration using IBM WebSphere Business Integration components.

WebSphere Commerce takes advantage of the IBM middleware platform (key components of which are integrated and ship with the product) to enable and accelerate the broadest, most encompassing range of integration capabilities on the market.

Figure 1-3 outlines the WebSphere Commerce cross channel product strategy.

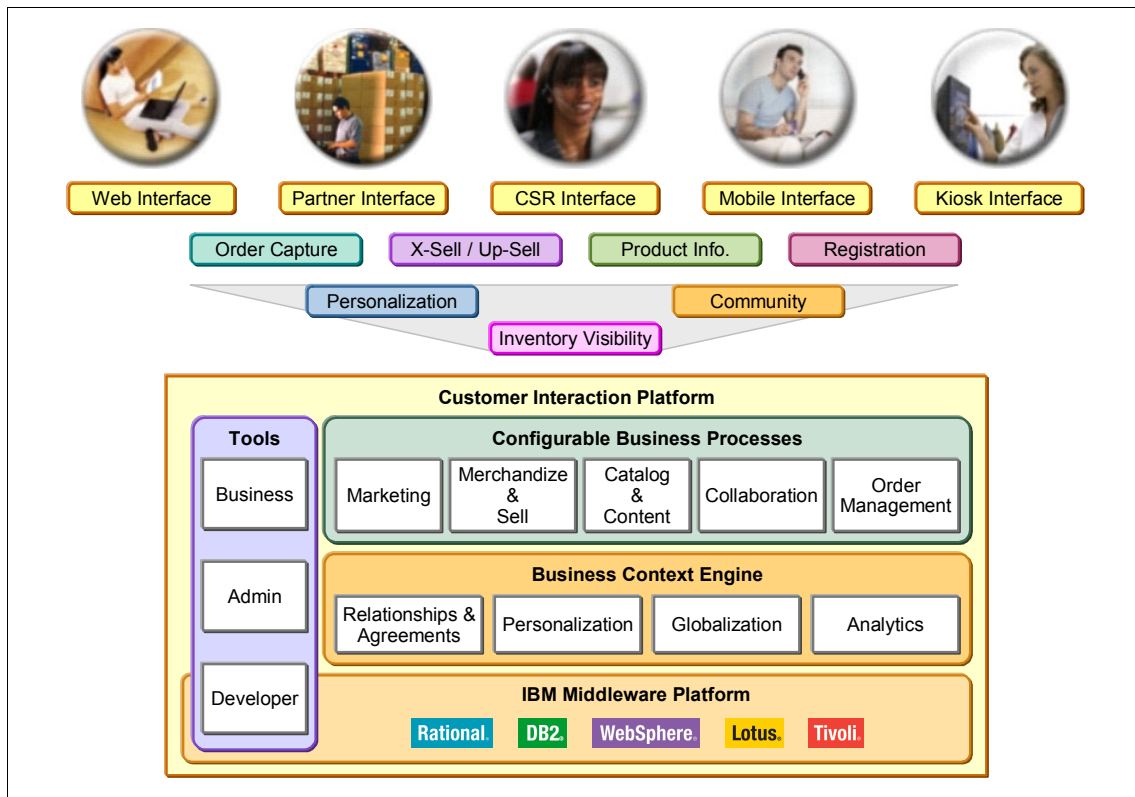


Figure 1-3 WebSphere Commerce cross channel product strategy

The WebSphere Commerce cross-channel product strategy includes the following features:

- ▶ Integrate customer-facing processes across touchpoints
WebSphere Portal and Pervasive help WebSphere Commerce enable a consistent experience and universal access to the data and processes that reside in enterprise-wide systems, across all of the touchpoints that the company supports and for all of types of users with whom a company interacts (such as employees, customers, and partners).
- ▶ Collaborate with employees, customers, and partners
Lotus® products help WebSphere Commerce enable real-time communication and collaboration (for example, live help, team selling, collaborative learning, and instant messaging) between and among employees, customers, and partners.

- ▶ Create a single, consolidated view of data and analytics
 DB2® helps WebSphere Commerce enable companies to take advantage of all of the various sources that store and track data and content about customers, products, orders, and so forth to deliver a single, consistent view of the customer to the company, as well as a single view of the company to the customer.
- ▶ Integrate with back-end and existing systems and connect to partners and suppliers
 WebSphere Business Integration products enable a range of integration options, whether a company needs simply point-to-point integration with a back-end system or more sophisticated process modeling and a complete demand-to-delivery integrated environment. In a demand-to-delivery environment, all business processes are linked horizontally and vertically across the value chain.
- ▶ Ensure the security and performance of the integrated environment
 IBM Tivoli® products manage the security and monitor and analyze performance of the integrated environment.
- ▶ Take advantage of open standards for maximum flexibility in integration
 Rational® products, industry-standard development environment and tools, Web Services, and adherence to open standards and protocols enable WebSphere Commerce to be deployed quickly and adapted to meet unique and changing needs and opportunities. In addition, these resources ensure that you can replace the IBM components of the system by custom or third-party add-ons.
- ▶ Integrate with store environments
 WebSphere Remote Server gives retailers components to define a robust infrastructure with an SOA approach using IBM Retail Integration Framework with your enterprise architecture model.

1.3.3 SOA for cross-channel integration

WebSphere Commerce fits into the IBM SOA Foundation Architecture using services components to build a cross channel solution. Figure 1-4 gives an overview of SOA Logical Model to use with WebSphere Commerce solutions.

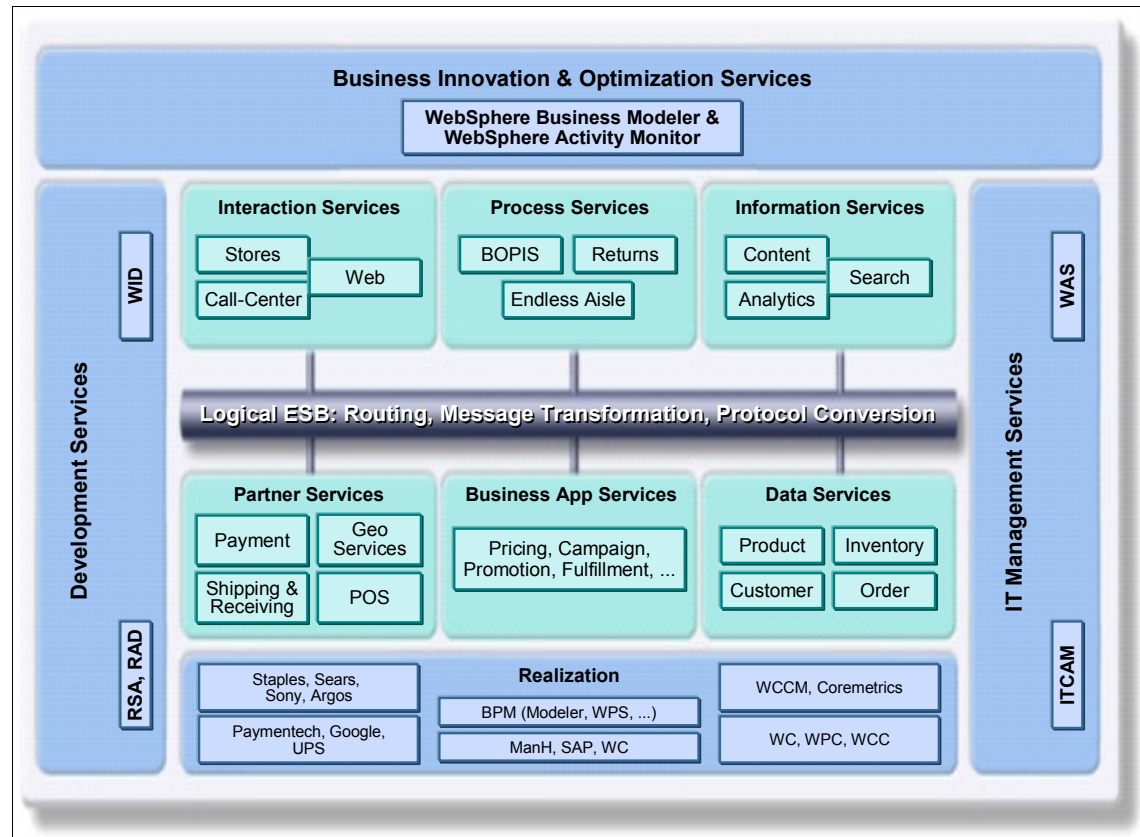


Figure 1-4 SOA for cross channel integration

Using SOA with WebSphere Commerce helps simplify integration between business processes and applications using standardized suggested components from retail industry models such as the SOA blueprint from Association for Retail Technology Standards (ARTS), IBM Retail Integration Framework, and Enterprise Architecture Frameworks.

For a specific scenario you must identify, model, and compare the following key assets with the previous reference frameworks:

- ▶ Key business processes
- ▶ Key applications
- ▶ Key roles and business units
- ▶ Key information
- ▶ Key channels

Then, you can follow an SOA-based approach using a top down or bottom up methodology, for example, using service-oriented modelling and architecture (SOMA) methodology to search for the following information:

- ▶ Specific common tasks called services
- ▶ Data integrity, quality, and completeness
- ▶ General business rules
- ▶ Business and IT events that you must measure

Finally, you can use WebSphere Commerce to define the following domain assets:

- ▶ A common model for product, customer, inventory, and orders to expose them as data services
- ▶ Common functionalities across channel such as Payments, Shipping and Receiving, and Geo Services
- ▶ Common scenarios using best practices that help to align IT with business objectives

1.4 Cross-channel configurations

This section describes how to integrate the following scenarios and three sample configurations to build a cross channel configuration using WebSphere Commerce:

- ▶ Buy or reserve online, pick up and pay in store
- ▶ Return or exchange in-store items bought online
- ▶ Buy in store when out-of-stock
- ▶ Offer and purchase additional items in store (targeted marketing)

Table 1-2 lists the three sample configurations.

Table 1-2 Cross-channel configurations

Configuration	Main WebSphere Commerce feature
WebSphere Commerce and DOM	<ul style="list-style-type: none"> ▶ Store Locator ▶ Cross-channel inventory visibility and allocation ▶ Order sourcing and fulfillment
WebSphere Commerce and BPM	<ul style="list-style-type: none"> ▶ Dynamic business modeling ▶ Configurable business rules ▶ Business Activity Monitoring (BAM)
WebSphere Commerce and Store	<ul style="list-style-type: none"> ▶ Point-of-Sale (POS) integration ▶ Endless aisles kiosks and mobile devices ▶ Cross-channel marketing and promotion

Figure 1-5 shows how the scenarios and configurations relate to each other.

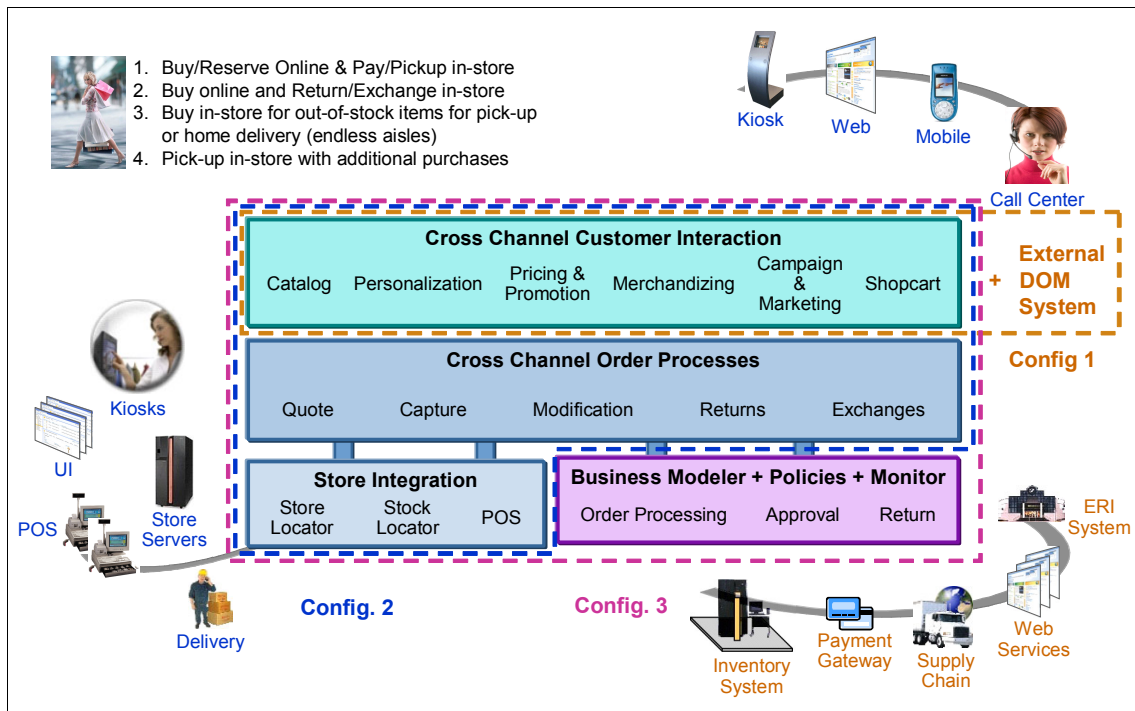


Figure 1-5 Scenarios and configurations

This book focuses on WebSphere Commerce with distributed order management (referred to as *DOM* throughout this book) and the mobile solution as the first

natural step on a roadmap to adopt a cross channel strategy. The goal is to build a continual state of business and IT alignment through a series of projects that support the highest-impact business scenarios for cross channel selling. You must plan carefully before moving to these scenarios. It is critical to answer the following questions:

- ▶ What business cross channel solution projects are implemented and in what sequence?
- ▶ What channels do we need to implement?
- ▶ What features will a channel deliver?
- ▶ Who can use this channel?
- ▶ Where to source the service or channel?
- ▶ How to manage the channel for performance and services levels?

You have a complete platform to deliver value using WebSphere Commerce features for cross channel solutions. We describe these features in the sections that follow.

1.4.1 WebSphere Commerce and DOM

In an enterprise where orders are managed by a dedicated back-end system, WebSphere Commerce offers the ability to integrate seamlessly with the order management system using SOA, which provides an integrated e-business solution that encompasses the entire order life cycle. In the integrated solution, WebSphere Commerce can provide rich front-ends for the sales channels, as well as robust functionalities in marketing, merchandising, personalization, order capture, and payment. In addition, the order management system can provide the back-end inventory and order management capabilities, plus integration with fulfillment and supplier networks as well as external services.

Figure 1-6 shows the high-level view of the integrated solution from a business perspective.

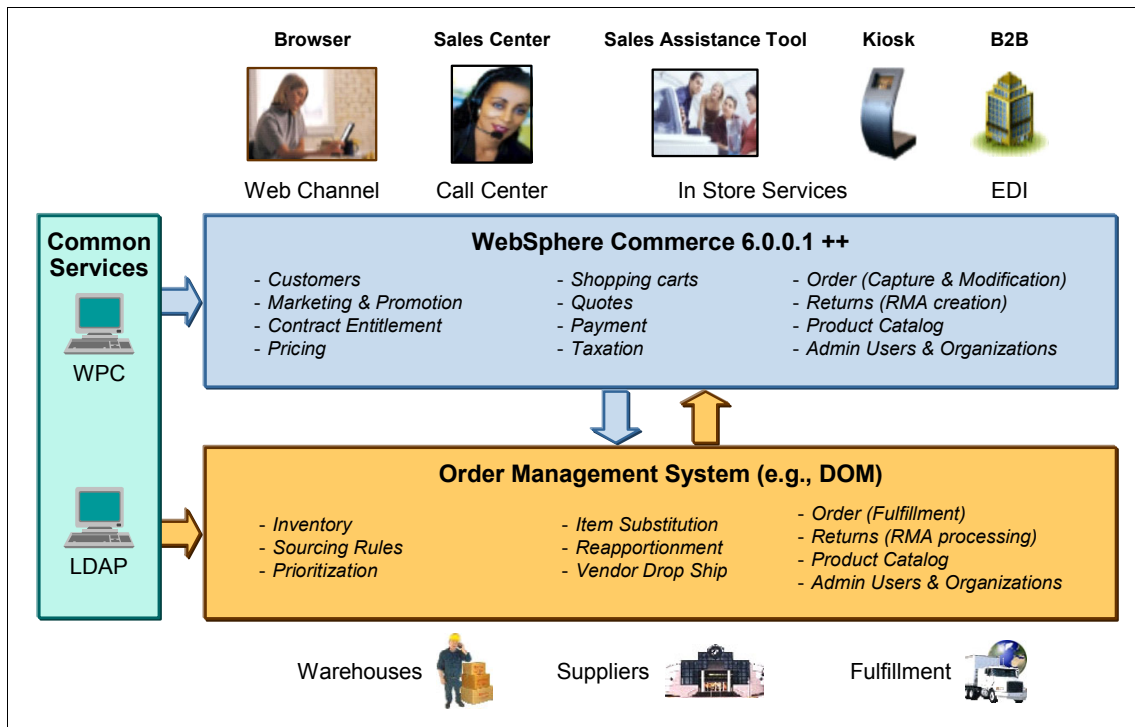


Figure 1-6 WebSphere Commerce integrated solution

Solution

WebSphere Commerce and DOM integration provides the following features as illustrated in the Figure 1-7 on page 20:

- ▶ Integrated business processes to support common customer usage scenarios, including cross channel inventory visibility; buy online, ship to, and pick up in store; and buy-online and return in store
- ▶ Open services interfaces and message flows for order transfers and updates, payment, and customer and inventory data over enterprise service bus (ESB) and SOA Foundation components
- ▶ Reference implementation that connects WebSphere Commerce with the DOM system (for example, Manhattan Associates, SAP)

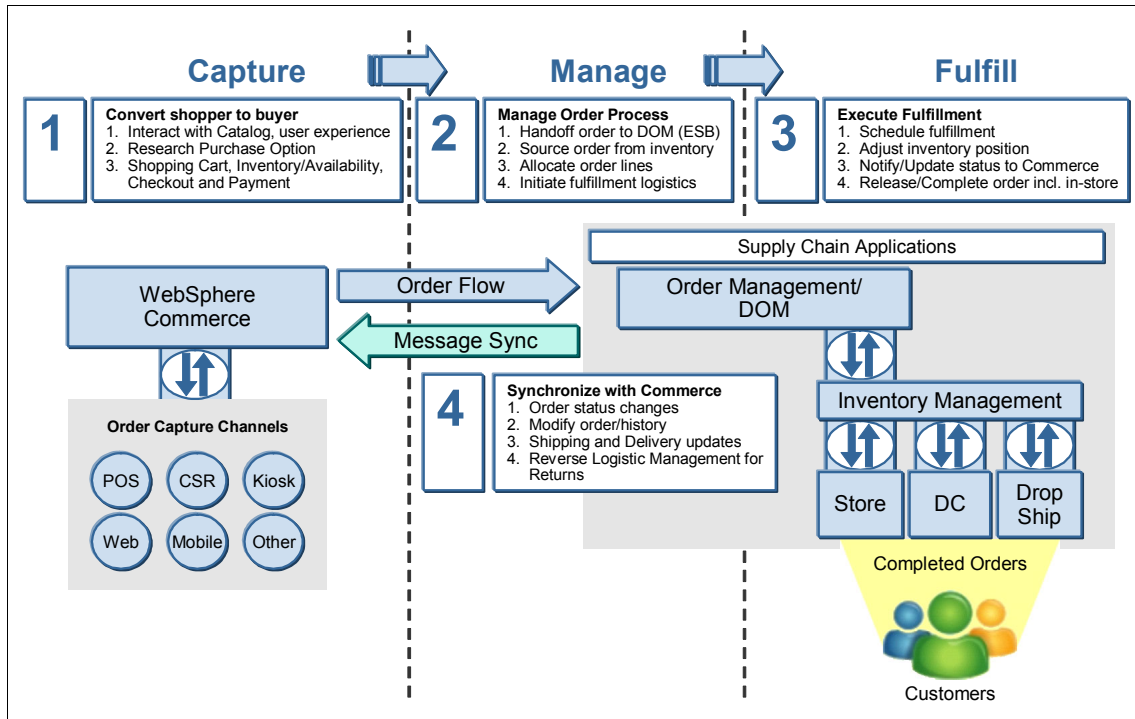


Figure 1-7 DOM cycle

Buy online, pick up in store is one of the scenarios on which we focus in this book. Most cross channel shoppers use the in-store pick up service because they do not want to pay for shipping or wait for the items to be delivered. When customers pick up an order in the store, they might also buy additional items, which increases the overall revenue.

To enable the buy online, pick up in store scenario, we use the following new and enhanced capabilities in WebSphere Commerce:

- ▶ **Stock Locator**, which allows shoppers to first look up in-store inventory availability information for any physical stores based on address and store attributes. DOM has visibility to inventory across stores and channels.
- ▶ **Enhanced WebSphere Commerce checkout** to capture not just the regular Web orders but also orders with the in-store pickup option. The enhancements are not limited only to the Web channel.
- ▶ **The in-store fulfillment component** to provide the fulfillment workflow for store associates to handle picking and packing these in-store pick up orders and to notify the shopper when the item is ready for pick up. WebSphere Commerce receives notifications for any status changes of the in-store pick up orders

from the in-store fulfillment component. Shoppers can also access real-time order status updates through WebSphere Commerce.

Example flow for buy online, pick up in store

Figure 1-8 illustrates the WebSphere Commerce and DOM integration flow for the buy online, pick up in store scenario (referred to as *BOPIS* in this book).

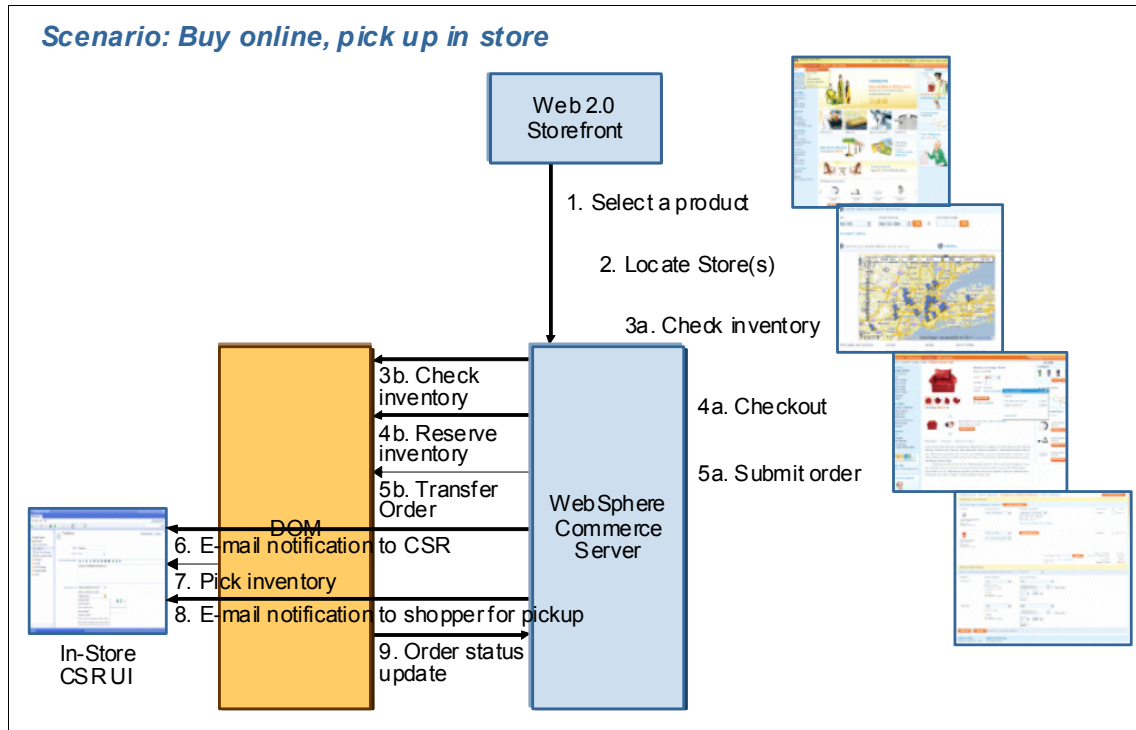


Figure 1-8 Buy online, pick up in store (BOPIS) flow

The BOPIS process works as follows:

1. A shopper browses the product pages and selects the product.
2. The shopper searches for and locates the store at which to pick up the product.
3. Commerce verifies its inventory cache for the inventory level of the product in the fulfillment center or centers. If the information is not cached in WebSphere Commerce, it calls DOM for the information.
4. The Commerce client user interface calls WebSphere Commerce to prepare the order for checkout. Then, Commerce calls DOM to reserve the inventory and to obtain the estimated ship date.

5. The WebSphere Commerce client user interface calls WebSphere Commerce to the submit order. WebSphere Commerce transfers the order to DOM. Depending on the configuration, WebSphere Commerce either transfers the order to DOM right away or waits until the payment is authorized.
6. Commerce sends an e-mail notification to CSR for the pick-up request.
7. DOM fulfills the order to the store.
8. WebSphere Commerce sends an e-mail notification to customer for pickup.
9. WebSphere Commerce pulls the order status from DOM.

1.4.2 WebSphere Commerce and BPM

Designed to transcend system, channels, and organizational boundaries, SOA can accelerate the speed and effectiveness of information sharing and process automation. A level of integration, coordination, and collaboration can also show positive effects throughout the value chain.

Using WebSphere Commerce and BPM together provides the following benefits:

- ▶ Increases overall business visibility and effectiveness throughout the entire value chain
- ▶ Increases customer satisfaction throughout all channels and touchpoints

Cross-channel scenario

As an example of using WebSphere Commerce with BPM, we describe how the Point-of-Sale (POS) solution retrieves tasks on a business process and how WebSphere Commerce tasks are pulled on an on-demand basis to the store. The POS solution using the WebSphere Remote Server infrastructure is connected to the store data center where WebSphere Process Server intercepts messages to determine the next action. In this example, the Association for Retail Technology Standards (ARTS) is coming from the store. The WebSphere Remote Server Central Site is using Web Services to connect with WebSphere Commerce.

Alternatively, an order is captured by a channel on WebSphere Commerce, and it can be pulled on demand to WebSphere Remote Server Central Site.

Figure 1-9 shows the IT infrastructure that we used to build this configuration.

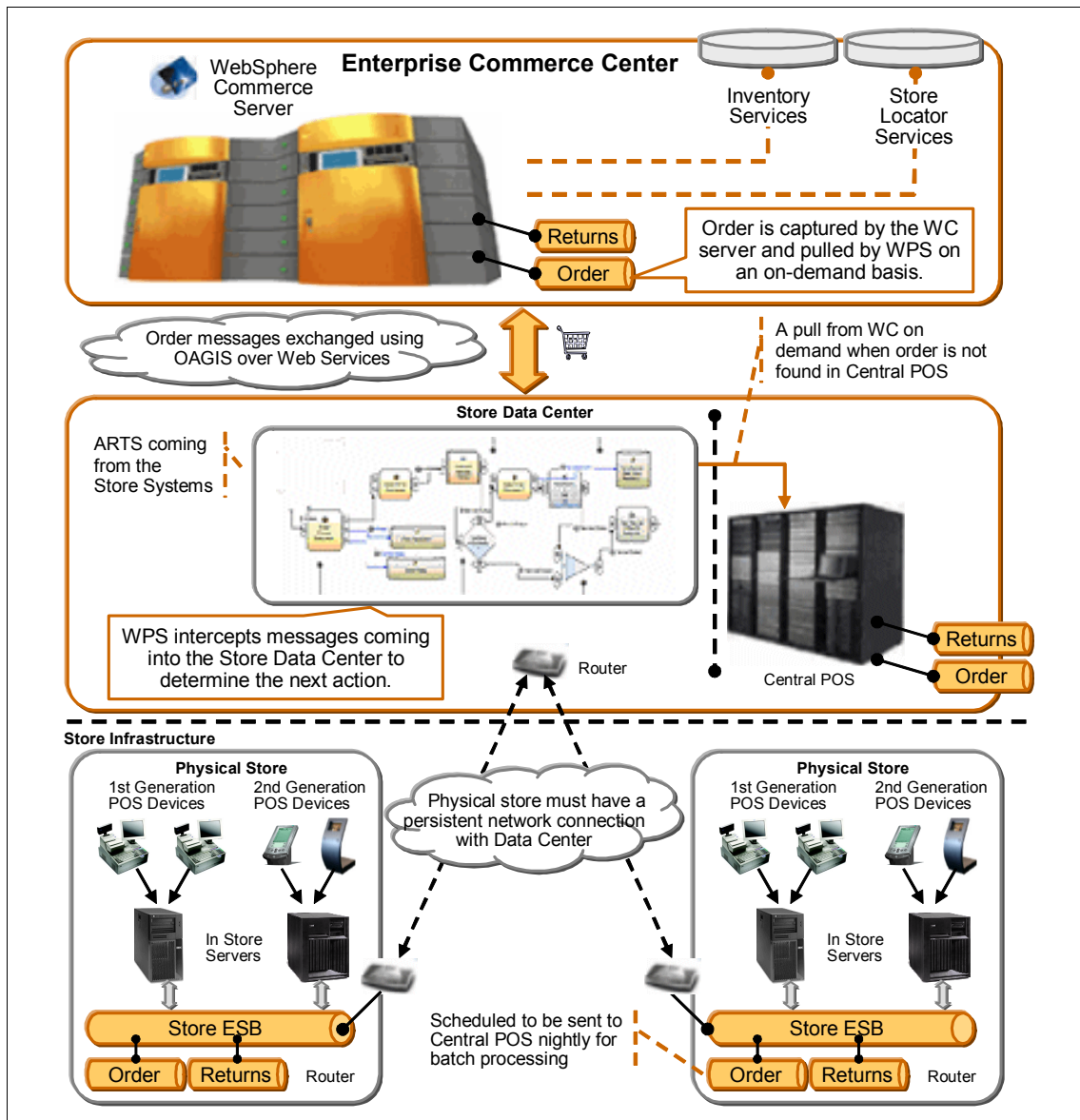


Figure 1-9 Using WebSphere Commerce and BPM together

1.4.3 WebSphere Commerce on store environments

Using WebSphere Commerce on store environments has the following benefits:

- ▶ Single view of customers and orders throughout all channel
- ▶ Seamless customer shopping experience throughout channels
- ▶ Cross-channel marketing and promotion through a CSR interface and in-store kiosks

Solution

Using the same approach that we used with the WebSphere Commerce and BPM scenario, we use a Retail Reference Model, IBM Retail Integration Framework, and an Enterprise Architecture Framework. Using a Retail Reference Model on a retail environment, you have the following scenarios:

- ▶ Store
- ▶ Distribution Center
- ▶ Central

Each scenario has its own services or tasks that must be accomplished, and those services are located as business application services.

On a logical model, you have a unique data model that uses a Retail Industry Standard such as ARTS. We have a domain that uses data services with master data management recommendations for product, customer, inventory, orders, and other elements of a cross channel solution environment.

IBM Retail Integration Framework uses these elements with POS integration. The main integration components for this SOA approach are WebSphere Enterprise Service Bus and WebSphere MQ.

Figure 1-10 illustrates this scenario using WebSphere Commerce throughout the value chain, including store integration processes.

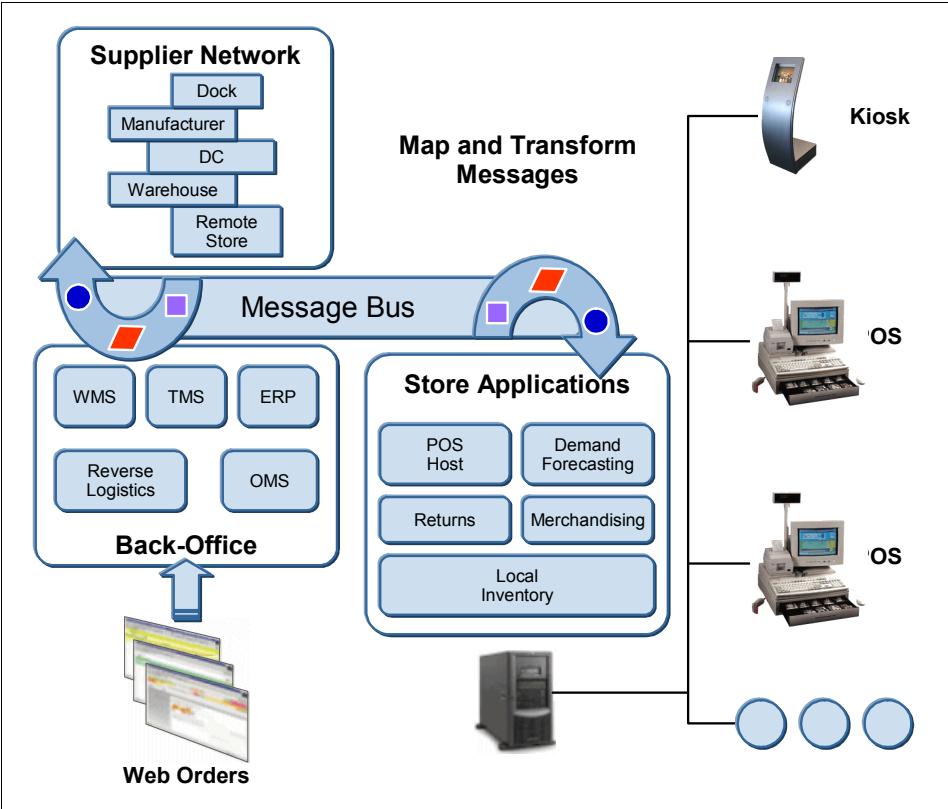


Figure 1-10 WebSphere Commerce throughout value chain including store processes

Table 1-3 lists the tools that you can use.

Table 1-3 WebSphere Commerce and store main components

Tool	Purpose
WebSphere Remote Server	A collection of IBM middleware products that is intended to be a base software stack for running Web applications within a store
WebSphere Enterprise Service Bus or WebSphere Message Broker	A software component which provides fundamental services for complex architectures using event-driven and standard-based-messaging-engine

Cross-channel scenario

We use open services interfaces and message flows for order transfer from WebSphere Commerce to POS applications through ARTS Remote Transaction Interface (RTI). POS Solution using WebSphere Remote Server infrastructure is connected to the store data center where ESB or MQ Broker intercept messages to dispatch them to WebSphere Commerce. Commands are used to implement basic functionality of WebSphere Process Server.

Alternatively, an order is captured by a channel on WebSphere Commerce, and it can be pulled by the ESB or MQ Broker to dispatch it to WebSphere Remote Server Central Site.

Figure 1-11 shows IT infrastructure to build this configuration.

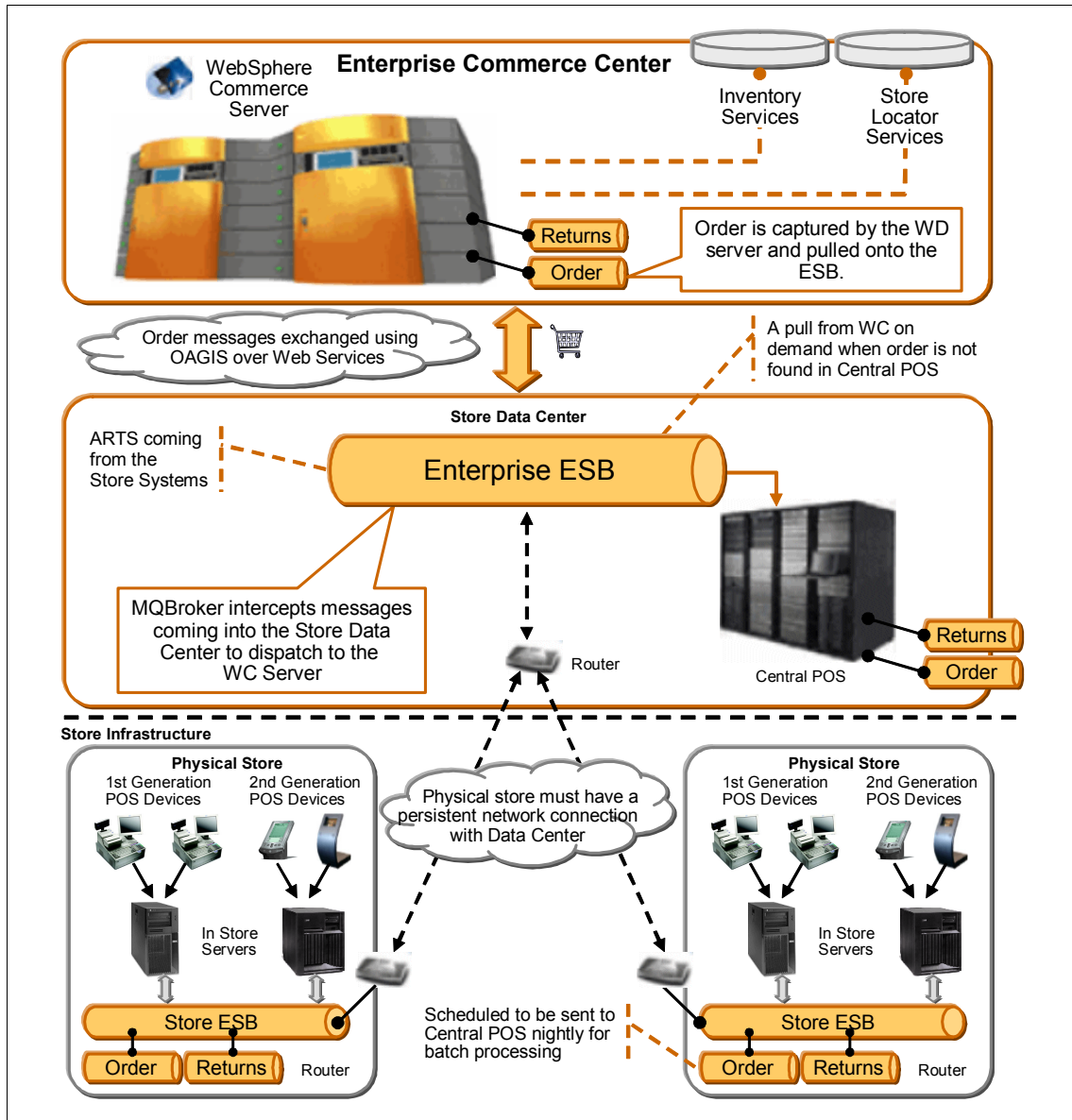


Figure 1-11 WebSphere Commerce on a store environment

1.5 Summary

WebSphere Commerce provides the basis to build a robust cross channel solution with expanded multichannel and customer centric features to deliver rich and contextual experience, including community interaction.



IBM WebSphere Commerce V7 features

This chapter describes the latest features and techniques of IBM WebSphere Commerce V7 and reviews the main features of IBM WebSphere Commerce V6 Feature Enhancement Pack 5.

The new and enhanced features in IBM WebSphere Commerce V7 are structured into general categories areas, which we discuss in the following topics:

- ▶ Enhanced Madisons Starter Store
- ▶ Social Commerce
- ▶ Marketing improvements
- ▶ Management Center enhancements
- ▶ Utilities
- ▶ Stack update

2.1 Enhanced Madisons Starter Store

In WebSphere Commerce V7, the Madisons Starter Store Web 2.0 business-to-consumer store, introduced in IBM WebSphere Commerce V6 Feature Enhancement Pack 2, is enhanced with new features and a general technology update. Furthermore, WebSphere Commerce V7 introduces a new business-to-business starter store, named *Elite*.

2.1.1 Madisons Starter Store

The first version of the Madisons Starter Store B2C store, highlighting rich Internet application (RIA) features, was introduced in Feature Enhancement Pack 2 of IBM WebSphere Commerce V6. Madisons Starter Store was enhanced with additional features in Feature Enhancement Pack 5 and is now further amended in IBM WebSphere Commerce V7.

Note: In addition to the specific feature and technical enhancements mentioned in this section, Madisons Starter Store is now a standard starter store and replaces ConsumerDirect as the primary starter store for use when developing new business-to-consumer stores.

In this section, we discuss the enhancements in the Madisons Starter Store that ships with IBM WebSphere Commerce V7, along with key enhancements from IBM WebSphere Commerce V6 Feature Enhancement Pack 5.

Buy online, pick up in store

The ability for shoppers to elect to pick up products in a brick-and-mortar store, rather than having the products shipped to their home or office, was introduced in IBM WebSphere Commerce V6 Feature Enhancement Pack 5. To use the *buy online, pick up in store* feature, referred to as *BOPIS* in this book, you need to enable the distributed order management (referred to as *DOM* throughout this book) functionality and provide a third-party inventory system, such as Manhattan Associates, to provide inventory information for the brick-and-mortar store to IBM WebSphere Commerce.

IBM WebSphere Commerce V7 enhances this model by allowing customers to use the built-in available to promise (ATP) or non-ATP inventory models to handle the inventory for the brick-and-mortar stores. As such, if customers want to use BOPIS but do not want to use an external inventory system, they can do this in IBM WebSphere Commerce V7.

The ability for shoppers to buy online and pick up the products in a store brings many benefits to both shoppers and merchants.

For shoppers, the main benefits include the ability to get the products immediately, rather than wait for the products to be picked, packed, and shipped, as well as the chance to inspect the product and return or exchange the products immediately if the product is not suitable. These advantages help increase customer confidence, especially for first-time shoppers.

For merchants, the main benefit is that it brings the customers into the physical store, increasing the possibility that the shopper buys extra products, for example, as the result of an impulse purchase.

Stock Locator

Introduced in IBM WebSphere Commerce V6 Feature Enhancement Pack 5 and enhanced in IBM WebSphere Commerce V7, the Madisons Starter Store Stock Locator allows customers to verify the online inventory status of a given product as well as to check the product's inventory status in any number of brick-and-mortar stores.

A shopper can keep a personalized list of stores that is reused whenever the shopper requests the inventory status of a product in a physical store.

As with the BOPIS feature, the shopper's visibility to the in-store availability of a product increases the chance that the shopper will visit a physical store with the associated up-sell potential.

Store Locator

The Store Locator allows shoppers the ability to search for brick-and-mortar stores within a specified geographical location, such as a city or zip code. The resulting list of stores shows the address and phone number of each store, providing the ability for shoppers to contact a given store directly. Furthermore, the store list also provides a quick way for shoppers to add physical stores to the personal store list, which is then used by the BOPIS and Stock Locator features.

DOM

While not strictly a Madisons Starter Store feature, the DOM integration capability that was introduced with IBM WebSphere Commerce V6 Feature Enhancement Pack 5 enables the aforementioned Stock Locator and BOPIS functions, warranting its mention here. The DOM feature provides exposure for IBM WebSphere Commerce to the inventory levels on a large number of inventory locations, typically brick-and-mortar stores, without having to replicate this information within the IBM WebSphere Commerce inventory tables. For more information about DOM and a sample integration scenario using IBM WebSphere

Message Broker, see Chapter 3, “Distributed order management integration” on page 77.

Improvements in the use of RIA features

The first version of the Madisons Starter Store in IBM WebSphere Commerce V6 Feature Enhancement Pack 2 introduced many RIA features, such as drag, add-to-cart without full screen refresh, and so forth. IBM WebSphere Commerce V7 improves several of these features:

- Removal of the automatic product Quick Info pop-up panels

In previous versions of the Madisons Starter Store, the product Quick Info panel opened automatically when the shopper moved the mouse pointer over a product thumbnail image on the category pages. This function had the tendency to impede the shopping flow and was considered intrusive to shoppers.

Now, in IBM WebSphere Commerce V7, when the shopper moves the mouse pointer over a product thumbnail image, only a small button displays, as shown in Figure 2-1. The product Quick Info view opens only if the shopper actively clicks this button.

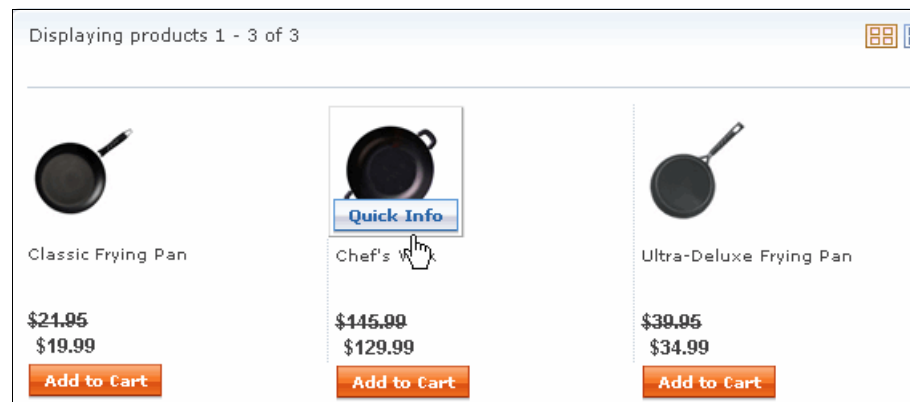


Figure 2-1 The Quick Info button

► Progress indicators

A usability issue that occurs when adding RIA functionality to a Web site is the difficulty of users to understand the progress of asynchronous actions. This issue is especially true for longer-running actions, such as adding products to the shopping cart.

To address this concern, the updated Madisons Starter Store introduces progress indicators for asynchronous actions. The indicator, the ubiquitous rotating circle ball animation, displays only when the shopper initiates an asynchronous action, as shown in Figure 2-2.

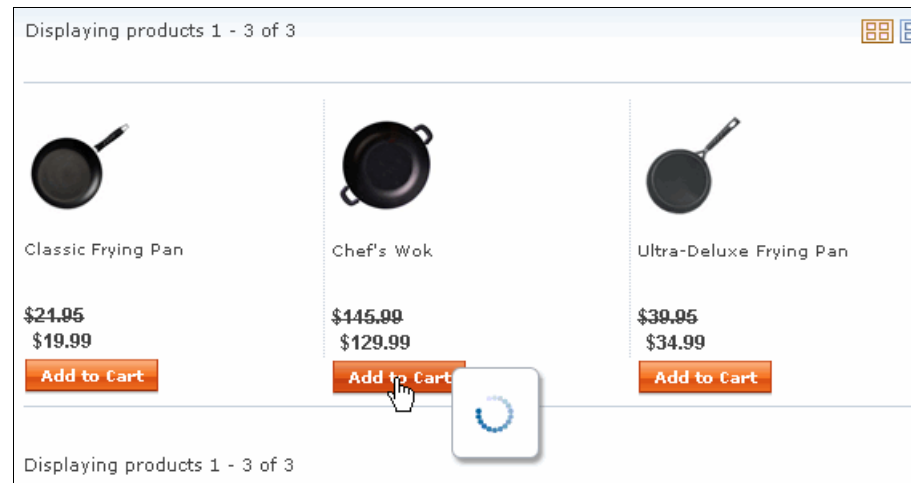


Figure 2-2 The progress indicator that displays when clicking Add to Cart

► Improved fast finder

The performance as well as the positioning of the product fast finder is improved in the IBM WebSphere Commerce V7 version of the Madisons Starter Store. In addition to the performance improvements inherent in the technology update (refer to "Dojo improvements" on page 38 for more information), the fast finder is optimized to handle very large categories. Instead of loading all products for the entire category during the initial page load, the fast finder loads only the first page of products and loads the remaining products in the background.

Figure 2-3 shows the product fast finder results.

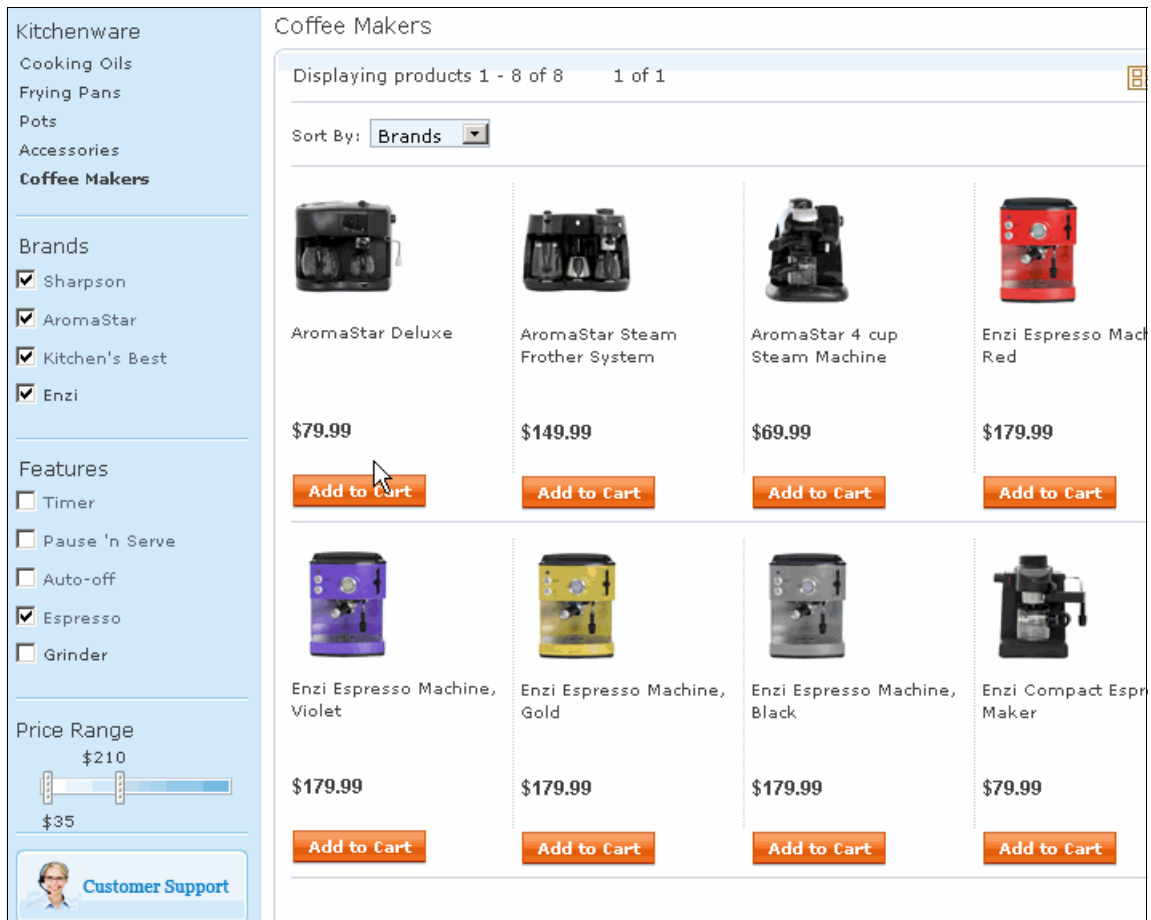


Figure 2-3 The product fast finder

Easier use of Shopping Cart, Personal Wish List, and Product Compare

In previous versions of the Madisons Starter Store, the accordion widget was used to implement a combined Shopping Cart, Personal Wish List, and Product Compare list. While this feature provided easy access to the contents, it took up a large amount of screen real estate. Furthermore, when shoppers attempted to use the drag functionality, the accordion compartments opened and closed in an almost uncontrollable manner, causing usability issues.

In IBM WebSphere Commerce V7, the Madisons Starter Store separates these three functions. The Shopping Cart is represented by an expandable mini shopping cart as shown in Figure 2-4.

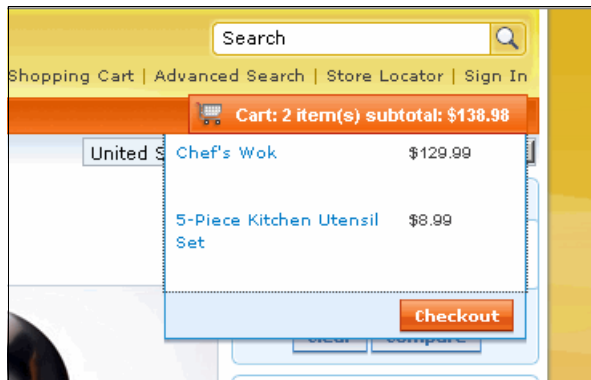


Figure 2-4 The mini shopping cart with two items in the cart

The Personal Wish List function is rolled back to the ConsumerDirect functionality in that the Personal Wish List is accessible only as a full-page function, as shown in Figure 2-5. The Personal Wish List feature is linked to from the footer of all pages in Madisons Starter Store.

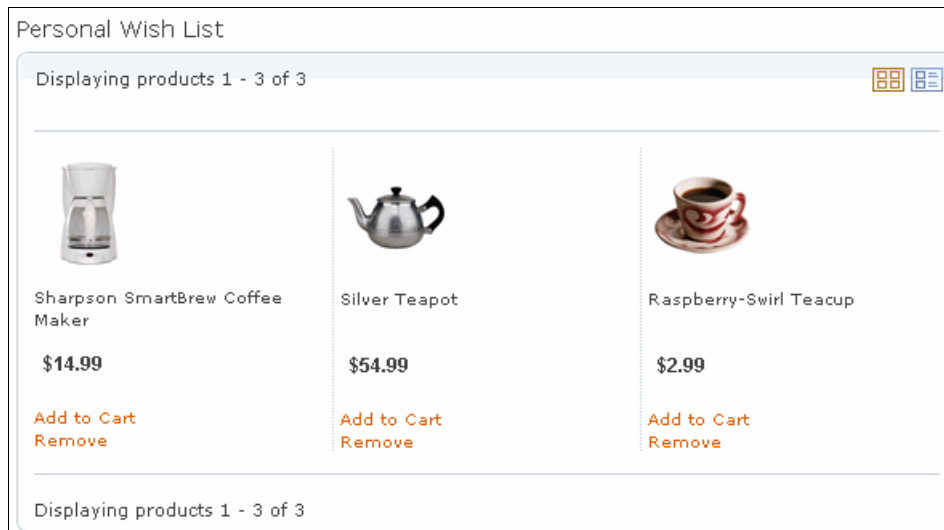


Figure 2-5 The Personal Wish List feature in Madisons Starter Store

The Product Compare area is located below the mini shopping cart as a fixed sized area that is always visible on the category and product pages, as shown in Figure 2-6.



Figure 2-6 The Product Compare area below the mini shopping cart

In-line editing of items in the Shopping Cart

In IBM WebSphere Commerce V7, the Madisons Starter Store introduces the ability for shoppers to modify product attributes in-place in the Shopping Cart. This function allows shoppers to change defining attributes, such as the size or color of an item, without having to remove the current SKU and add the new SKU with other attributes.

Coupon wallet

In IBM WebSphere Commerce V7, the Madisons Starter Store introduces the concept of a *coupon wallet*. This function provides registered shoppers with the ability to view the coupons for which they qualify and to apply for coupons directly from the wallet upon checkout, without having to remember and manually enter coupon codes.

Shopping Cart pagination

Pagination is added to the Shopping Cart. This feature is useful for limiting the size of the page when the shopping cart list of items is very large, which is typically the case for grocery merchants, where shopping carts with many items are commonplace. With the pagination feature, shoppers can change the page size using simple JavaServer Pages (JSP) customization. The default size is 20 items.

New pay in store payment method

A new payment method that allows customers to pay in the store rather than giving payment information online is now available. This method is available for shoppers who elect to buy online and to pick up the products in a physical store.

In addition to simplifying the order process, the shopper does not need to enter credit card information online with this feature. This feature is also beneficial in markets where cash and check payments are the predominant payment methods. It also supports merchants who require that payment capture is handled through in-store terminals.

Improved accessibility

In IBM WebSphere Commerce V7, the Madisons Starter Store includes several changes to the underlying HTML that can increase the accessibility of the site:

- Improved screen reader support

The W3C Web Accessibility Initiative (WAI) traditionally has discouraged all use of JavaScript in accessible Web sites (c.f. Checkpoint 6.1 of the Web Content Accessibility Guidelines (WCAG) 1.0).

In recognition that this guideline is not sustainable with the proliferation of RIA Web sites, W3C WAI published the *Accessible RIA Suite* (WAI-ARIA). The current version of this document at the time of writing is Working Draft 4. WAI-ARIA prescribes methods for annotating the HTML that is used in an RIA such that the resulting Web site is accessible to people with disabilities.

The use of Dojo widgets is changed to conform to WAI-ARIA. Specifically, all refresh areas are amended with WAI-ARIA defined attributes. These attributes provide information to screen readers about the role and importance of each area.

All refresh areas within Madisons Starter Store are defined as *live regions*, which is the WAI-ARIA term for dynamically refreshed areas of a Web page. These regions are defined as being *polite*, which means that the shopper is informed about changes of the content in that area only when the shopper is done with the current task.

- Use of semantic HTML

The starter stores of previous versions of IBM WebSphere Commerce were heavily dependant on using tables for layout. The Madisons Starter Store that ships with IBM WebSphere Commerce V7 uses Cascading Style Sheet (CSS) technology with HTML `div` tags instead, which results in JavaServer Pages (JSPs) that are easier to customize and maintain and provides a better separation of content from layout.

Store archives for extended sites models

In addition to the basic composite store archive for publishing a stand-alone Madisons Starter Store, IBM WebSphere Commerce V7 also delivers storefront asset stores for the extended sites models. The storefront asset stores allow store developers to publish a customer-facing store based on the business logic and views of the Madisons Starter Store B2C store.

Dojo improvements

The Dojo Toolkit, which is used in the Madisons Starter Store, includes a number of technical changes that provide for RIA features:

- ▶ Previous versions of the Madisons Starter Store used the Dojo Toolkit Version 0.4.1. The Madisons starter store now uses the Dojo Toolkit Version 1.3.1, which includes a number of functional and performance improvements.
- ▶ The Madisons Starter Store archive now includes the Dojo Toolkit, which eliminates the need to download the toolkit from the Dojo Web site.
- ▶ The JavaScript library used in Madisons Starter Store is customized to contain only the functions that are needed within the Madisons Starter Store, which results in a substantially reduced download size.
- ▶ The following new Dojo widgets are added to the IBM WebSphere Commerce specific libraries:

- WCDialog

This widget is a customized version of the standard Dojo `dijit.Dialog`, which is used in the Madisons Starter Store to display the contents of the shopping cart when the shopper moves the mouse pointer over the mini shopping cart area, as shown in Figure 2-7.

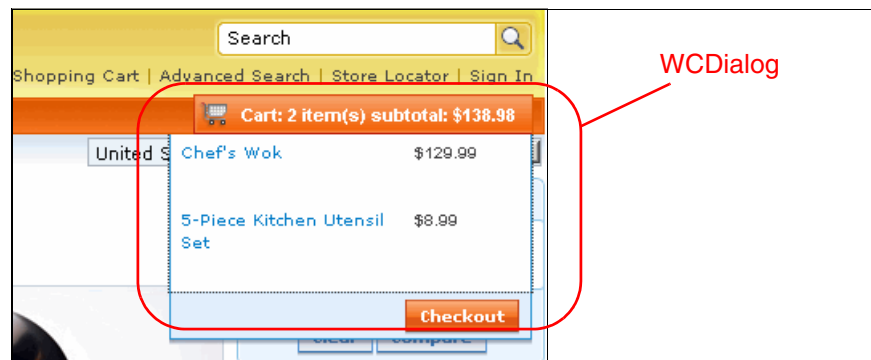


Figure 2-7 The WCDialog widget

- WCDropDownButton

This widget implements a drop-down type menu where the menu items are hidden until the shopper moves the mouse pointer over the menu name. WCDropDownButton uses the WCMMenu widget to display the drop-down menu. Figure 2-8 illustrates how the WCDropDownButton widget is used in Madisons Starter Store to generate the top-level category buttons in the header of every page.

- WCMMenu

This widget is an extension of the standard `dijit.Menu` widget, providing framework support for menus such as the top categories on the home page of the Madisons Starter Store. Figure 2-8 shows the use of WCMMenu in the Madisons Starter Store to create the submenu with the second-level categories whenever the shopper moves the mouse pointer over one of the top-level categories in the header. Each of the second-level categories (Cooking Oils, Frying Pans, Pots, Accessories, and Coffee Makers in Figure 2-8) is implemented as a `dijit.MenuItem` instance.

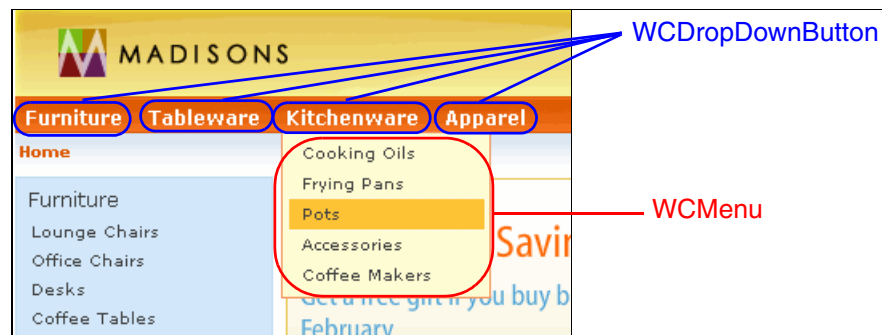


Figure 2-8 The WCDropDownButton and WCMMenu widgets

- The Dojo parser allows manually specified IDs of DOM nodes.

The standard behavior for Dojo upon loading a page is for the parser to traverse the entire DOM to search for embedded Dojo widgets. This process can be time consuming and, thus, can result in a noticeable delay from the time a page is initially rendered by the shopper's browser until any Dojo widgets finish rendering and become responsive.

To address this issue, the Madisons Starter Store in IBM WebSphere Commerce V7 uses a different approach in which the page developers manually specify the IDs of the DOM nodes that contain Dojo widgets to the Dojo parser. This change results in a much more efficient page initialization process and, thus, a faster response time for shoppers.

New Change Flow options

The Change Flow pages within the WebSphere Commerce Accelerator for the Madisons Starter Store in IBM WebSphere Commerce V7 are amended with functions to switch on and off specific RIA features within the customer-facing store. The use of RIA features in the following areas can be controlled through Change Flow pages, as shown in Figure 2-9:

- ▶ **Product Quick Info**

This switch controls whether the shopper can display product details in a pop-up panel directly from any product thumbnail image, as described in “Improvements in the use of RIA features” on page 32.

- ▶ **Ajax add to Shopping Cart**

This feature controls whether Ajax is to be used for adding products to the Shopping Cart. If this flow feature is enabled, then the shopper does not leave the current page if a product is added to the Shopping Cart.

If disabled, the functionality is similar to the functionality within the ConsumerDirect starter store, whereby adding a product to the Shopping Cart is a synchronous action that results in displaying the Shopping Cart page.

- ▶ **Ajax checkout**

This flow feature determines whether the changes in the Shopping Cart are applied asynchronously as they are made or whether the shopper has to click a button after making a change in order to apply the changes.

- ▶ **Ajax My Account**

This flow feature determines whether the changes on the My Account pages are applied asynchronously as they are made or whether the shopper has to click a button after making a change in order to apply the changes.

- ▶ **Product drag**

This flow feature controls whether the shopper can add products to the Shopping Cart by dragging the product thumbnail images to the mini shopping cart. If enabled, shoppers can also drag products to the compare area in a similar way.

Note: This feature is disabled automatically if both the Compare zone (under Catalog) and mini shopping cart (under Orders) flow features are disabled.

Customer Interactions

Select the Web 2.0 features that customers can use in your store:

Display product details in the Product Quick Info pop-up window.

☒ Product Quick Info

Add items to the shopping cart without leaving the current page.

☒ AJAX add to shopping cart

Automatically apply changes made to the shopping cart.

☒ AJAX checkout

Automatically apply changes made on the My Account page.

☒ AJAX My Account

Add items to the shopping cart, or compare zone, by using the mouse.

Note: If the mini shopping cart and compare zone features are disabled; then this

☒ Product drag-and-drop

Figure 2-9 Web 2.0 Change Flow features

New <wcf:url> tag

The <wcf:url> tag generates links on the shopping pages and is added to the wcf tag library. This tag replaces the <c:url> tag from the JavaServer Pages Standard Tag Library (JSTL) and reduces the amount of effort that is needed to implement search engine optimization (SEO) URLs in the store. Example 2-1 shows an example of the use of the standard JSTL <c:url> tag.

Example 2-1 Use of the standard <c:url> JSTL tag

```
<c:url var="myURL" value="CategoryOnlyResultDisplayView">
  <c:param name="storeId" value="{WCPParam.storeId}" />
  <c:param name="langId" value="{langId}" />
  <c:param name="catalogId" value="{WCPParam.catalogId}" />
  <c:param name="categoryId" value="{WCPParam.categoryId}" />
</c:url>
```

Example 2-2 shows how to change the code from Example 2-1 to use the specialized `<wcf:url>` tag. The changes are highlighted in bold.

Example 2-2 Use of the specialized IBM WebSphere Commerce `<wcf:url>` tag

```
<wcf:url var="myURL" value="CategoryOnlyResultDisplayView">
  <wcf:param name="storeId" value="{WCPParam.storeId}" />
  <wcf:param name="langId" value="{langId}" />
  <wcf:param name="catalogId" value="{WCPParam.catalogId}" />
  <wcf:param name="categoryId" value="{WCPParam.categoryId}" />
</wcf:url>
```

Functionality between the JSTL `<c:url>` tag and the IBM WebSphere Commerce V7 `<wcf:url>` tag is different in the following ways:

- ▶ The `<wcf:url>` tag generates the full path, including the protocol. For example, instead of generating the following path:
`CategoryOnlyDisplayView?storeId=10001&langId=-1&catalogId=10001&categoryId=10001`

The `<wcf:url>` tag generates this path:

```
http://www.redbooks.ibm.com/webapp/wcs/stores/servlet/CategoryOnlyDisplayView?storeId=10001&langId=-1&catalogId=10001&categoryId=10001
```

Note: This example assumes that the SEO URLMapper is not enabled.

- ▶ If the SEO URLMapper is enabled, the `<wcf:url>` tag automatically generates a URL that matches the current rules that are defined in `URLMapper.xml`. The example in the previous bullet list item thus results in the following path:

```
http://www.redbooks.ibm.com/webapp/wcs/stores/servlet/CategoryOnlyDisplay1_10001_-1_10001_10001
```

- ▶ The `<wcf:url>` tag prefixes the URL automatically with the appropriate protocol, depending on the `https` setting in the struts configuration file, thus eliminating the necessity of a round trip to the server. Figure 2-10 shows a comparison between using the server to redirect to an SSL connection versus generating the links with the HTTPS protocol to begin with. As Figure 2-10 illustrates, generating the links with the HTTPS protocol embedded saves a round trip for each request.

Furthermore, if the page does not require SSL, the `<wcf:url>` tag prefixes the link using the Hypertext Transfer Protocol (HTTP) protocol, thereby switching back to the more efficient non-SSL HTTP protocol.

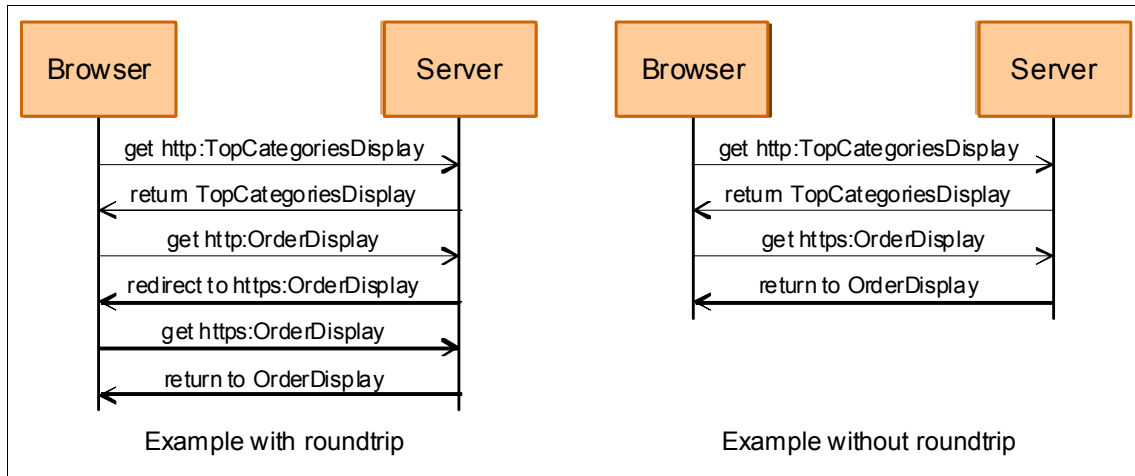


Figure 2-10 Comparison of using the server to redirect to SSL versus generating SSL links

Increased use of services instead of data beans

The coverage of component services is expanded in IBM WebSphere Commerce V7. As such, the use of the `<wcf:getData>` tag instead of the `<wcbase:useBean>` tag in the Madisons Starter Store starter store is increased as well. Example 2-3 shows an example of retrieving order details using the `<wcbase:useBean>` tag.

Example 2-3 Using the `<wcbase:useBean>` to retrieve order details with a data bean

```

<wcbase:useBean
  classname="com.ibm.commerce.order.beans.OrderDataBean"
  id="orderBean"
  scope="page">

  <c:set property="orderId" value="${WCPParam.orderId}"
    target="${orderBean}" />

</wcbase:useBean>
  
```

Example 2-4 shows how to implement the code from Example 2-3 using component services.

Example 2-4 Using `<wcf:getData>` to retrieve order details through component services

```

<wcf:getData
  type="bom.ibm.commerce.order.facade.datatypes.OrderType"
  var="order"
  expressionBuilder="findByOrderId">
  
```

```

        <wcf:param name="accessProfile" value="IBM_Details" />
        <wcf:param name="orderId" value="{WCParam.orderId}" />

</wcf:getData>

```

Comparing the code in Example 2-3 with that in Example 2-4 shows that the switch to using component services is not necessarily for simplicity of the code. Instead, the use of component service provides a better abstraction, allowing the component services to retrieve data in a way that can be more efficient for the task at hand. The move to using component services is a continuation of the service-oriented architecture (SOA) adoption within IBM WebSphere Commerce.

Note: Although Example 2-3 and Example 2-4 show equivalent methods for retrieving data, there are areas in which using the `<wcf:getData>` tag is the only way to retrieve the data, for example for some of the precision marketing functions in IBM WebSphere Commerce V7. (For more information, refer to 2.3.4, “Precision marketing” on page 65.) Some of the new capabilities within this area are not exposed as classic data beans and, thus, cannot be accessed using the `<wcbase:useBean>` tag.

Using JavaScript for Ajax code directly in JSPs

When the first Web 2.0 reference store was released in IBM WebSphere Commerce V6 Feature Enhancement Pack 2, it introduced a number of JSP tags. These tags simplified the generation of the required JavaScript for setting up Ajax services, render contexts and refresh controllers by generating the required JavaScript at run time.

Although these JSP tags provided a convenient way to generate the required JavaScript for refresh controllers, they made it more cumbersome to change the underlying libraries or to add extra parameters to the JavaScript. Thus, the Madisons Starter Store JSPs in IBM WebSphere Commerce V7 use JavaScript directly in the JSPs for declaring these objects and areas.

Table 2-1 compares the previous JSP tags to the new JavaScript declarations with a short description of what each tag does.

Table 2-1 Comparison of previous JSP tags to new Java Script declarations

Feature Enhancement Pack 2	IBM WebSphere Commerce V7	Description
wcf:declareService	wc.service.declare	Defines the endpoint for an asynchronous service call.

Feature Enhancement Pack 2	IBM WebSphere Commerce V7	Description
wcf:declareRenderContext	wc.service.declareContext	Defines an area that can be refreshed as the result of an Ajax call.
wcf:declareRefreshController	wc.service.declareRefreshController	Defines a controller that determines what to refresh when responses are returned from the server.

Although JSP tags are available, do not use them, because they generate code that is compatible only with Dojo Version 0.4. Use the JavaScript declarations for new Ajax development.

Additional languages

In addition to the standard 10 group one languages, the Madisons Starter Store is translated to the following languages:

- ▶ Polish (pl_PL)
- ▶ Romanian (ro_RO)
- ▶ Russian (ru_RU)

For reference, the following are the 10 group one languages with the associated locale names:

- ▶ French (fr_FR)
- ▶ German (de_DE)
- ▶ Italian (it_IT)
- ▶ Japanese (ja_JP)
- ▶ Korean (ko_KR)
- ▶ Portuguese (pt_BR)
- ▶ Simplified Chinese (zh_CN)
- ▶ Spanish (es_ES)
- ▶ Traditional Chinese (zh_TW)
- ▶ US English (en_US)

Functions not included in the Madisons Starter Store

The version of Madisons Starter Store that ships with IBM WebSphere Commerce V7 does not include the following functions:

- ▶ Gift Center

There is no Gift Center store archive for Madisons Starter Store, so you need to customize any gift registry functionality.

- ▶ Customer Care

Customer Care is the ability for shoppers to initiate a chat session with a customer service representative (CSR), and it allows the CSR to retrieve the shopper's profile and the shopping cart information.

The version of the Madisons Starter Store that ships does not support a Customer Care feature. If you require this feature, you need to customize the Madisons starter store.

- ▶ Attribute dictionary

The attribute dictionary is a concept that was introduced in IBM WebSphere Commerce V6 Feature Enhancement Pack 2.

In the traditional attribute model in IBM WebSphere Commerce, attributes are defined on the product level, even for attributes that are seemingly identical, such as the color or size of different shirts in the catalog.

The attribute dictionary allows for sharing common attributes across products. With the attribute dictionary, the catalog manager defines the color or size attributes and its values only once for the entire catalog and can simply assign those attributes to products for which the attributes apply.

Although the attribute dictionary feature is available in IBM WebSphere Commerce V7, the Madisons Starter Store does not take advantage of the attribute dictionary. Thus, this feature is not supported in Madisons Starter Store.

- ▶ Auctions

Because Madisons Starter Store is a business-to-consumer store, auctions are not supported.

- ▶ Request for Quote (RFQ)

Because Madisons Starter Store is a business-to-consumer store, RFQ is not supported.

2.1.2 Enhanced business-to-business starter store

In addition to the traditional business-to-business starter stores, IBM WebSphere Commerce V7 includes a new enhanced starter store, called *Elite*.

Elite adds all of the RIA features from Madisons Starter Store to the advanced B2B direct starter store, such as drag, quick product information, direct editing in the shopping cart, product fast finder, and so forth, as well as some of the technical enhancements that are added to Madisons Starter Store in IBM WebSphere Commerce V7, such as semantic HTML, accessibility features, additional languages, and SEO optimized URLs.

2.1.3 Madisons Mobile Starter Store add-on store archive

Recognizing the growth in the mobile e-commerce market, IBM is delivering an add-on store archive, `MadisonsMobile.sar`, with IBM WebSphere Commerce V7 to publish on top of the Madisons Starter Store. An *add-on store archive* is a store archive that does not have the assets that are necessary to create a complete store. Instead, the add-on store archive is intended to add capabilities to an existing store. The Gift Center store archives for IBM WebSphere Commerce V6 are examples of add-on store archives for the ConsumerDirect starter store.

The `MadisonsMobile.sar` store archive adds mobile capabilities to the Madisons Starter Store, which we describe in the following sections.

Mobile pages

The main visible feature of the Madisons Mobile Starter Store add-on store archive is that it contains a set of pages that provide much of the existing Madisons Starter Store functionality, catered to the constraints of a mobile device. The major differences between the regular Madisons Starter Store pages and the Madisons Mobile Starter Store pages are that the mobile pages are designed to fit within the display of a Tier-1 Smartphone, for example a device with a display resolution of 240 by 320 pixels. Furthermore, the use of RIA features is minimized to better match the capabilities of mobile devices, such as the limits on JavaScript capabilities and on the size of documents that are sent to the device.

Some pages are changed to reduce the amount of data that is necessary for shoppers to input in order to complete various tasks. For example, the number of fields for registration on the Madisons Mobile Starter Store store is reduced to the following fields, as shown in Figure 2-11:

- ▶ Logon ID
- ▶ Password (and Verify Password)
- ▶ First Name
- ▶ Last Name
- ▶ E-mail

Register

To create an account please register below. All fields marked with an (*) are required.

*Logon ID

*Password (6 characters minimum with 1 numeric)

*Verify Password

First Name


*Last Name

*E-mail

Figure 2-11 User registration form for Madisons Mobile Starter Store

These reduced fields are in contrast to the following fields, which are required for a standard Madisons Starter Store registration process, as shown in Figure 2-12:

- ▶ Logon ID
- ▶ Password (and Verify password)
- ▶ First Name
- ▶ Last Name
- ▶ Street address
- ▶ City
- ▶ Country/Region
- ▶ State/Province
- ▶ ZIP code/Postal code
- ▶ E-mail
- ▶ Phone number
- ▶ Store specials option
- ▶ Preferred language
- ▶ Preferred currency
- ▶ Gender
- ▶ Birthday
- ▶ Mobile Phone Number
- ▶ Short Message Service (SMS) preferences
- ▶ Remember Me preference

 Please Register Below

* denotes required fields

* Logon ID:		* Verify password:	
* Password:		* Last Name:	
First Name:		* City:	
* Street address:			
		* State/Province:	
* Country/Region:	Afghanistan		
* ZIP code/Postal code:			
* E-mail:		Phone number:	
<input type="checkbox"/> Send me e-mails about store specials.			
Preferred language:		Preferred currency:	
United States English		US Dollar	
Gender:			
select one			
Birthday:			
Year	Month	Date	
----	--	--	Privacy Policy
Mobile Phone Number:		Mobile Phone Number:	
Country:		+93	
Afghanistan			
<input type="checkbox"/> Send SMS notifications to mobile phone			
<input type="checkbox"/> Send SMS promotions to mobile phone			
<input type="checkbox"/> Remember Me		Example: +91 1234567890	
<input type="button" value="Submit"/> <input type="button" value="Cancel"/>			

Figure 2-12 User registration form for Madisons Starter Store

Finally, the pages shipped in the Madisons Mobile Starter Store store are tested to conform to W3C's mobileOK Basic Tests 1.0, which verify a subset of the W3C Mobile Web Best Practices 1.0 specification. For more information, refer to:

<http://www.w3.org/TR/mobile-bp/>

Mobile access keys

As part of making the pages more accessible to mobile devices, access keys are set up for the functions listed in Table 2-2. These store-wide access keys are defined for quick access to some of the central functions.

Table 2-2 Store-wide access keys for Madisons Mobile Starter Store

Access Key	Function
1	Home
2	Shopping Cart
3	Search
4	Product Comparison
5	Store Locator
6	Change Language/Currency
7	Personal Wish List
8	My Account
9	Contact Us

SMS foundation support

IBM WebSphere Commerce V7 adds support for SMS through integration with third-party gateways. The following new Java EE Connector Architecture (JCA) resource adapters ship with IBM WebSphere Commerce V7:

► JCASMS-HTTP

This resource adapter implements a simple HTTP-based SMS interface, used by many SMS gateway providers. The resource adapter accepts any number of name-value pairs, defining various aspects of the message, such as the recipient's phone number and the message type, as well as protocol data, such as authentication keys and the target URL.

The transport that is predefined to map to this resource adapter defines the most common subset of name-value pairs and provides the ability to amend this set with any number of extra parameters.

► JCASMS-WS

This resource adapter implements part 4 of the Parlay X 3.0 specification. Part 4 defines the specifications for a Web Services based SMS gateway. As such, any merchant that has access to a Parlay X compliant SMS gateway can use the transport that is defined for this resource adapter to send text messages to mobile devices.

Note: The transports and resource adapters for SMS messaging is strictly not part of the Madisons Mobile Starter Store. We describe it in this section, because the Madisons Mobile Starter Store is the store that is most likely to make use of these adapters.

The SMS adapters and transports are part of the IBM WebSphere Commerce V7 foundation.

For more details, refer to “Using SMS functionality for mobile marketing” on page 193.

SMS order status updates

One use of the SMS adapters described in “SMS foundation support” on page 50 is for sending order status messages to a mobile device. This feature allows shoppers to get real-time order status information, which is useful if the shopper is not close to a personal computer.

Functions not included in the Madisons Mobile Starter Store

The following functions are omitted from the mobile pages on Madisons Mobile Starter Store compared to the regular Madisons Starter Store storefront:

- Merchandising associations
- Coupon wallet in My Account
- Quick Checkout profile in My Account
- Address Book in My Account
- Checkout and Ship to Home or Office

Note: Only the shipping options are not available in Madisons Mobile Starter Store. By using Madisons Mobile Starter Store, shoppers can still checkout through the buy online and pick up in store delivery option. Refer to 5.4, “Buy on mobile device and select shipping address” on page 221 for an example of adding shipping to the checkout pages for Madisons Mobile Starter Store.

These functions are omitted mostly due to the limitations of the mobile device. For example, managing an address book is not as easy from a mobile device as

from a personal computer. There are no technical limitations to adding any of these features.

2.2 Social Commerce

Social Commerce is a term coined to describe the convergence of e-commerce with social networking on the Web. Merchants are investing in Social Commerce features for their e-commerce site to create and nurture the community around the brand and Web site. With the increase of use of social networks on the Web, creating a virtual community around the site communicates to the shoppers that the merchant respects shoppers' opinions and helps solidify the brand loyalty.

Versions of the Madisons starter store prior to IBM WebSphere Commerce V7 were constrained to the RIA aspects of Web 2.0 and did not address the social aspects of e-commerce. With IBM WebSphere Commerce V7, however, the following Social Commerce features are incorporated into the product:

- ▶ Sharing on social networks
- ▶ Ratings and reviews
- ▶ Product and category blogs
- ▶ Photo and video galleries
- ▶ Social profile

We describe each of these features in the following sections. We also include a technical discussion regarding implementation details in 2.2.6, "Social Commerce integration" on page 58.

2.2.1 Sharing on social networks

The simplest Social Commerce feature is the addition of shortcuts for users to share a given page (for example, a product or a category page) on a *social network*. Figure 2-13 shows an example of four social network sharing buttons.



Figure 2-13 Social network sharing buttons on a product page

Figure 2-13 shows the following social network sharing buttons, listed left-to-right:

- Digg

Digg is a community site where users vote on Web pages that are submitted by other users. Many people use the highest ranked pages (determined by the number of *diggs* that a page receives) to get inspiration for visiting new sites. To visit this site, go to:

<http://www.digg.com/>

► Delicious

Delicious provides public bookmark functionality. Users can add their own sites to their public profile, assign tags to each bookmark, and publish the bookmarks for other users to see. Other users can then choose to save any of those bookmarks in their own profile.

The popularity of a bookmark is determined by the number of users who save a given bookmark. Delicious is another service that is used to get inspiration for visiting new sites. To visit this site, go to:

<http://www.delicious.com/>

► Facebook

Facebook is a popular social networking Web site. Facebook was rated the most popular social networking site in the world by *complete.com* in February 2009, closely followed by MySpace. Clicking the Facebook icon allows the user to add a status update the current page or to send and view private messages on Facebook. To visit this site, go to:

<http://www.facebook.com/>

► Google Bookmarks

Google Bookmarks is a service similar to Delicious, with the notable difference that bookmarks stored with Google Bookmarks are not shared with other users. So, rather than providing a bookmark sharing service, Google Bookmarks allow users to store their bookmarks in a central location and, with a number of tools such as a Mozilla Firefox plug-in, the users can synchronize bookmarks with their browser's bookmark list. To visit this site, go to:

<http://www.google.com/>

2.2.2 Ratings and reviews

Product *ratings* and *reviews* provide the ability for shoppers to share information about the quality and use of the individual products. Ironically, shoppers are more prone to trust the opinions of complete strangers than a trusted brand, as long as these complete strangers are thought to be independent on the merchant or brand owner. As such, adding ratings and reviews can have a great impact on the buying decisions of the shoppers.

Ratings and reviews are commonly hosted by third-party providers. These providers can then offer dedicated review capabilities to filter out reviews with inappropriate content. The third-party provider chosen for the sample review feature in IBM WebSphere Commerce V7 is BazaarVoice, which you can find at:

<http://www.bazaarvoice.com/>

Figure 2-14 shows an example excerpt from the reviews for the Red Leather Roll Arm Chaise product within Madisons Starter Store.


[Description](#) [Attachments](#) [Blogs](#) [Photos](#) [Reviews](#)

[Red Leather Roll Arm Chaise Review Entries](#) [Refresh](#)


[Register](#) to create a new review or [Sign In](#) in if you are already a member

Overall Rating : ★ ★ ★ ★ ★ 4.1 of 5


[Highest rating](#) [Newest post](#) ▼ Sort by:




real head turner this one..
★★★★★
real head turner this one.. take my word for this and just bring it home
by: [April](#) 2/27/09 5:03 AM




Take a Look at this
★★★★★
This classic set offers plush comfortable seating and is perfect for color-rich decors
by: [Chris](#) 2/27/09 5:03 AM




I just Love this
★★★★★
Contemporary upholstered armchair for the modern household. The best seat in the house, blending style and comfort
by: [Natasha](#) 2/27/09 5:00 AM



Ultimate
★★★★★
Ultimate comfort and featuring easy-care leather.
by: [kirsten](#) 2/27/09 4:56 AM



Just buy this one
★★★★★
It is just brilliant and would have given it 10 if i could
by: [David](#) 2/27/09 4:55 AM



Brilliant
★★★★★
Worth buying any time any day.. Very comfortable and stylish
by: [jill](#) 2/27/09 4:55 AM

[Highest rating](#) [Newest post](#) ▼ Sort by:

Figure 2-14 Ratings and reviews for the Red Leather Roll Arm Chaise

2.2.3 Product and category blogs

Somewhat similar to reviews, *product and category blogs* also show regular shoppers' opinions about products, but the format is more free-form. The blog functions allow other shoppers to comment on blog entries and, thereby, create more of a discussion forum. Product blogs are typically more centered around how individual shoppers use the products, as opposed to rating the product.

Depending on the market segment, blogs can also be more than just entries about a specific product. For example, a home improvement site can have blogs where regular visitors blog about home improvement plans, project progress with links to specific products, or even wish lists. This sharing of information can be very powerful in the creation of a community.

IBM WebSphere Commerce V7 ships with adapters that support blogs for the following products:

- Pluck

<http://www.pluck.com/>

- IBM Lotus Connections V2.5

<http://www.ibm.com/software/lotus/products/connections/>

Figure 2-15 shows an example of a single blog entry for the Lounge Chairs category in Madisons Starter Store.

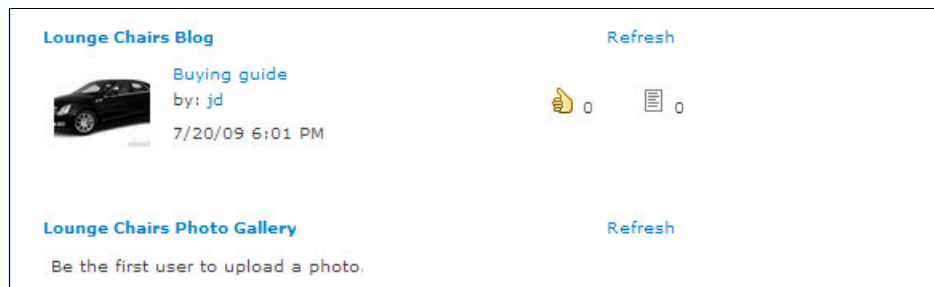


Figure 2-15 Blog and Photo Gallery entries for the Lounge Chairs category

2.2.4 Photo and video galleries

Although the reviews and blogs are primarily text-based, the *photo and video gallery*, allows customers to show action shots of the products. These photographs can be either humorous depictions of the products in special contexts, or more serious photos and videos showing how some shoppers have used the product in a special way. Merchants can also use the photo and video gallery to highlight special features or to show assembly instructions to shoppers.

IBM WebSphere Commerce V7 ships with adapters that support galleries for the following products:

- ▶ Pluck
<http://www.pluck.com/>
- ▶ IBM Lotus Connections V2.5
<http://www.ibm.com/software/lotus/products/connections/>

Figure 2-15 on page 56 shows the placement of the photo gallery for a category in Madisons Starter Store, although shoppers have not yet uploaded photos to that category.

2.2.5 Social profile

To support the more discussion-oriented components of Social Commerce (for example, blogs and galleries), customers need to create a public, or *social*, profile. While the profile that is set up as part of the standard IBM WebSphere Commerce registration flow is very personal and is expected to contain precise personal information about the shopper, the social profile typically contains only information that the shopper wants to share. The user name is often without resemblance to the shopper's real name, and the standard fields exposed in IBM WebSphere Commerce V7 for a social profile are limited to the following fields:

- ▶ Screen Name
- ▶ E-mail Address
- ▶ Hometown
- ▶ Birthday
- ▶ Gender
- ▶ Interests

All of these fields, apart from the screen name, are optional, allowing shoppers to share only as much information as they are comfortable. Figure 2-16 shows the registration form for creating a social profile.

☒ **Create an Online Community profile:**

Screen Name:

Email Address:

Hometown:

Birthday:

yyyy/mm/dd

Gender:

Interests:

The max character length is 150

Figure 2-16 Creating a social profile

2.2.6 Social Commerce integration

As mentioned in the previous sections, Social Commerce features in IBM WebSphere Commerce V7 require integration with either third-party service providers or applications. In this section, we describe the integration method for adding Social Commerce features to IBM WebSphere Commerce V7.

The chosen integration method is using a REST-based API integrated using WebSphere sMash V1.1 from Project Zero:

<http://www.projectzero.com/>

The WebSphere sMash server acts as a mediator between the client and the service provider, providing a generic Representational State Transfer (REST) API. IBM WebSphere Commerce V7 ships with sMash adapters for BazaarVoice, Pluck and IBM Lotus Connections V2.5 that then transform the generic REST requests to the specific API for each of the supported service providers.

Figure 2-17 illustrates the integration architecture for the Social Commerce implementation in IBM WebSphere Commerce V7.

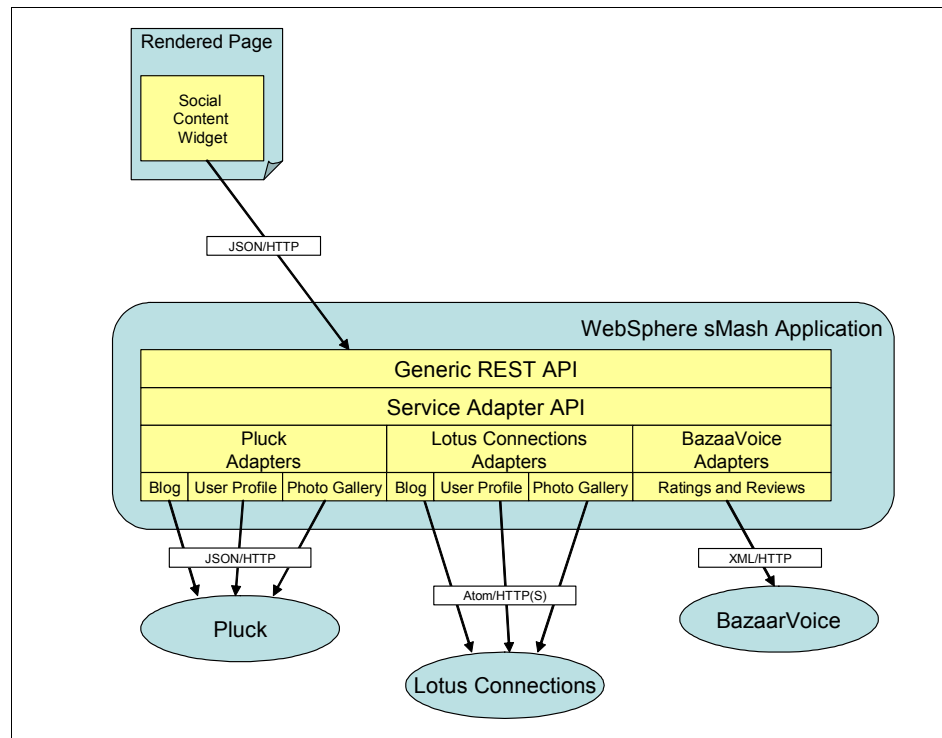


Figure 2-17 The Social Commerce integration architecture through WebSphere sMash

For more information about building WebSphere sMash applications, refer to *Building Dynamic Ajax Applications Using WebSphere Feature Pack for Web 2.0*, SG24-7635.

2.3 Marketing improvements

IBM WebSphere Commerce V7 introduces many new marketing features. You now get more control on how you identify the customer throughout multiple channels, create customer segments based on demographics, and target appropriate promotions based on customer browsing behavior. These controls greatly enhance conversion rates because shoppers see only highly personalized and relevant marketing content based on their current and past actions.

2.3.1 New promotion types

Promotions enable you to offer customers incentives to purchase. WebSphere Commerce supports numerous types of promotions, such as price promotions including simple discounts, merchandise specials such as gifts with purchase and buy-one-get-one, and service promotions including reduced shipping costs.

Business users create and manage promotions using the Promotions tool in the Management Center.

IBM WebSphere Commerce V7 offers the following new promotion types:

- ▶ **Order level: Amount off shipping**
This promotion type offers an amount off the shipping charges when the total value of the order meets or exceeds a specified amount (for example, the customer receives \$10 off shipping on orders over \$100).
- ▶ **Product level: Fixed shipping discount**
This promotion type offers an amount off the shipping charges when the order contains a specified number of catalog entries or a specified amount is spent on the catalog entries (for example, buy two or more “Villagois” Table Glasses and get free shipping).
- ▶ **Category level: Fixed shipping discount**
This promotion type offers shipping at a fixed price when the customer purchases a specified number of items or spends a specified amount on catalog entries from category *X* in one order (for example, buy two or more items from the Table Glasses category and get free shipping).
- ▶ **Multiple items percent discount**
This promotion type offers a percentage off a specific combination of items when the order contains all the items (for example, buy four “Villagois” Table Glasses and four “Villagois” Wine Glasses and get 10% off all these items).

Important: To avoid unexpected results at the storefront, item A and item B must be mutually exclusive. In other words, item A cannot include item B, and item B cannot include item A. For example, if item A is any item from the Hat category and item B is a fishing hat (a SKU within the Hat category), then this promotion is not valid because item A includes item B. This rule also applies if you have additional items in the purchase condition (items A, B, C, and D).

2.3.2 Deprecated promotion adjustments

The following promotion types are deprecated in IBM WebSphere Commerce V7 and are not available in promotion tool:

- ▶ Percent off volume discount
- ▶ Amount off volume discount

2.3.3 New promotion features

The IBM WebSphere Commerce promotion engine is enhanced and now supports the features that we describe in this section.

Maximum amount for discounts

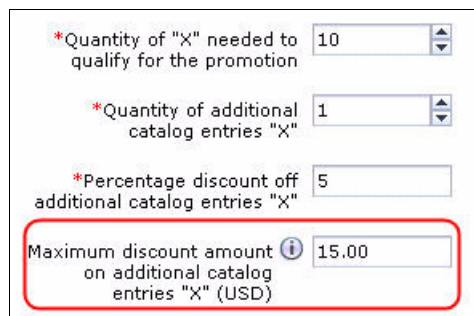
If the selling price is high, the discount amount after calculation can be large. IBM WebSphere Commerce V7 allows you to fix a cap on the discount amount (for example, 20% off greeting cards but only up to \$2). You can place the cap for both order-level promotion (as shown in Figure 2-18) and product-level promotion (as shown in Figure 2-19).



* Minimum Order Purchase (USD)	* Percentage Discount on Order (%)	Maximum Discount Amount (USD)
100.00	5	30.00
1,000.00	10	300.00
5,000.00	15	1,500.00

1 to 3 of 3

Figure 2-18 Maximum discount amount on order-level promotion



*Quantity of "X" needed to qualify for the promotion: 10

*Quantity of additional catalog entries "X": 1

*Percentage discount off additional catalog entries "X": 5

Maximum discount amount on additional catalog entries "X" (USD): 15.00

Figure 2-19 Maximum discount amount on product-level promotion

Payment type based

IBM WebSphere Commerce V7 allows you to reward a customer for using a particular payment type (for example, 10% off an order when paid with a store credit card). This feature requires a single payment type per order, as shown in Figure 2-20. Orders paid using multiple payment types do not qualify for the promotion.

Purchase Condition and Reward

*Minimum purchase condition

* Minimum Order Purchase (USD) 100.00

* Amount Off (USD) 10.00

1 to 1 of 1

*Target payment type VISA Credit Card

Figure 2-20 Full order must be paid for with the required payment type

Choice of free gift

IBM WebSphere Commerce V7 allows you to offer free gifts to customers from a list of options, providing you the opportunity to offer multiple free gifts. Figure 2-21 shows how marketing managers can create a list of products in the Management Center that can be offered as free gifts to the shoppers.

Free gift options

☐ Free gifts are automatically added to shopping cart

☒ Customer can choose free gifts from a list

Find and Add

*Free gift catalog entries

* Type	* Code	Name
	KIAC-0101	Measuring Spoons
	KIAC-0301	Rolling Pin
	KIAC-0401	Mortar and Pestle
	KIAC-0501	Spoons and Spatulas
	KIAC-0601	5-Piece Kitchen Utensil Set

1 to 5 of 5

*Number of free gift selections customer can choose 2




Figure 2-21 Choosing a free gift option

When shopping on the Web site, the user is presented with a list of promotional products (Figure 2-22) that are configured by the marketing manager using the promotion tool. The customer can choose a product from this list.

Select your free gifts close X

☒ I would like the following free gifts (choose up to 2 gifts)

☐ I do not want any free gifts

<input type="checkbox"/>		5-Piece Kitchen Utensil Set Everyday kitchen utensils in a container.	\$9.06 \$8.99
<input type="checkbox"/>		Spoons and Spatulas A handy mix of cooking spoons and spatulas.	\$6.06 \$4.99
<input type="checkbox"/>		Measuring Spoons Set of five measuring spoons.	\$9.06 \$7.99

..

Apply **Cancel**

Figure 2-22 List of promotional products

Attribute filtering

While creating promotions, you can use product attributes to filter catalog entries to which the promotion applies (for example, 10% off red shirts as shown in Figure 2-23).

Attributes for catalog entries ⓘ

* Attribute Name	* Data Type	* Matching Rule	* Value
Trim Color	Text	Attribute value car ...	Black-and-white che

Figure 2-23 Filter catalog entries based on attribute

Best deal calculation

The promotion engine supports an optional best deal feature that compares promotions *that have the same priority* automatically and applies them in the sequence that offers the lowest overall purchase price. The best deal feature runs only when:

- ▶ A customer's order qualifies for multiple promotions in the same promotion group
- ▶ Multiple promotions have the same priority

Important: Because of performance reasons, even after you enable the best deal feature, aim to control which promotions are applied first to a customer's order by assigning those promotions a higher priority. When you cannot predict the best deal in advance, assign the conflicting promotions the same priority, and let the best deal feature do the math.

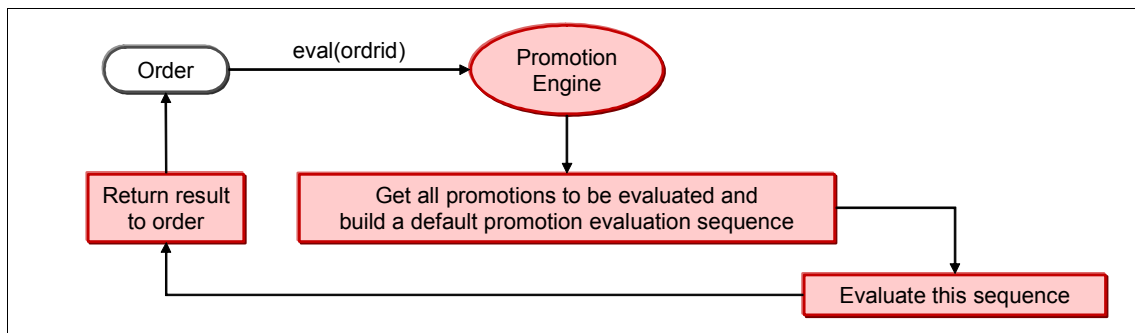


Figure 2-24 Traditional promotion evaluation

In previous versions of IBM WebSphere Commerce, when multiple promotions have the same priority, the promotion engine builds up a default sequence of promotions and applies them, which might or might not offer the best discounts for the customer. However, in IBM WebSphere Commerce V7, when you enable the best deal feature, the promotion engine builds all sequence combinations (subject to a defined threshold) for promotions that have same priority. These sequences are then evaluated, and the best result is selected, as shown in Figure 2-25.

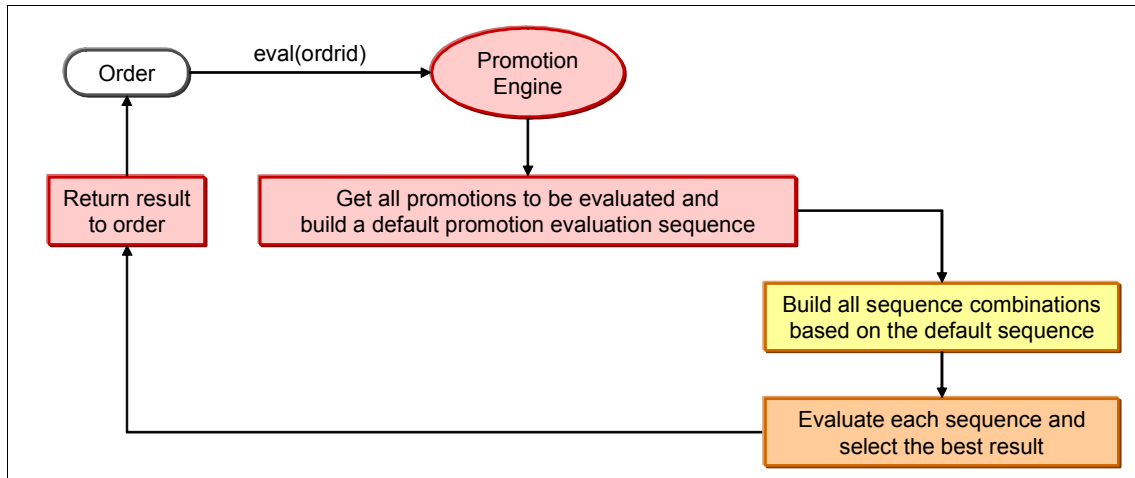


Figure 2-25 Best-deal support based on threshold

2.3.4 Precision marketing

Precision marketing refers to targeting content towards user based on current and past actions. Marketing content can be targeted towards both registered and guest users. The Management Center provides a rich UI that you can use to enable precision marketing.

Note: *Precision marketing* works using the *personalization ID*, which uniquely identifies a user and allows WebSphere Commerce to present the user with personalized content when the user interacts with the business, throughout the business life cycle. By default, the *persistent session* and *personalization ID* features are disabled. If not enabled, the customer segment and order purchase history targets can only target a customer based on the current *member ID* of that customer.

Figure 2-26 shows an Web activity diagram from the Management Center that displays the commonly used terms in precision marketing.



Figure 2-26 Terminology used in precision marketing

New targets

A *target* defines which customers experience marketing activities. When a customer reaches a target in the activity flow, the customer is evaluated against the target criteria. For example, the criteria for a Purchase History target might be that the customer has placed exactly five orders. Targets are typically based on a customer's behavior and segmentation. If you do not include targets in an activity, then the activity applies to all customers.

IBM WebSphere Commerce V7 introduces the following targets:

► Catalog Browsing Behavior

You can use Catalog Browsing Behavior to target customers who have browsed specific parts of the store catalog while shopping on the Web site over time. For example, if a customer has repeatedly browsed certain categories, then that customer is a prime target for advertisements or promotions that are related to those categories.

► External Site Referral (not for Dialog activities)

You can use External Site Referral to target customers who entered the Web site from a link on a specific external site.

► Social Commerce Participation

You can use Social Commerce Participation to target customers who have participated in social commerce on the Web site a specified number of times. Customers participate in Social Commerce when they do any of the following activities on your site:

- Post a product review or comment, or rate a product
- Post a blog entry or comment, or rate a blog entry
- Upload a photo or video

► Current Page (not for Dialog activities)

You can use the Current Page to target customers who are currently viewing a specific page of the store. For example, this target might be a display page for a specific part of the catalog or a search results page following a specific keyword search.

► Online Behavior

You can use Online Behavior to target customers whose recorded behavior while shopping on your site over time meets certain criteria. The recorded behavior of customers can provide important clues about their interests. You can use these clues to personalize marketing messages. For example:

- Customers who have searched the site using the keyword “television” within the last week.
- Customers who have visited store pages with page URLs that contain certain data.

► Cookie Contents (not for Dialog activities)

You can use Cookie Contents to target customers whose computers have a cookie from your site that contains certain data (for example, a zip code that indicates that the customer lives in a certain geographical area). This target is useful for targeting guest users who have not logged in but who have sufficiently browsed the site providing useful bits of information.

► Day and Time

You can use the Day and Time to specify the days of the week and times of day that an activity is active. If an activity uses the Branch element and has more than one path, use this target to set a different schedule for each path.

New actions

An *action* defines what to do based on the previous sequence of triggers and (optionally) targets in the activity flow. An action is a step to perform as part of the activity flow. In a Web activity, an action typically displays something in an e-Marketing Spot. In a Dialog activity, an action can send the customer an e-mail or text message or can add the customer to a customer segment.

IBM WebSphere Commerce V7 introduces the following new actions:

- ▶ Recommend Promotion
- ▶ Add to or Remove from Customer Segment
- ▶ Display Recently Viewed

New branch type

A *branch* allows you to define an alternate action or more than one course of action in an activity flow. IBM WebSphere Commerce V7 introduces the following new branch type:

- ▶ All paths for which the customer qualifies

New Dialog activity

IBM WebSphere Commerce V7 introduces *Dialog activity*. It is similar to a Web activity, but it allows multi-step interaction that is initiated by specific customer action. Dialog activity extends beyond e-Marketing Spots, because it allows you to track customer behavior over a period of time and to take actions accordingly. Figure 2-27 shows a sample Dialog activity.

Note: If Dialog activity is waiting (for example on a wait trigger), the activity does not continue until that path is complete.

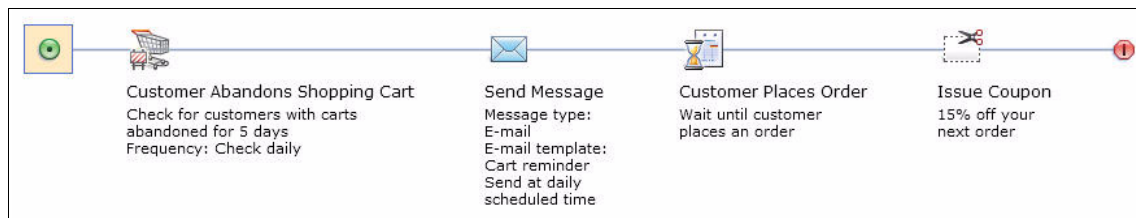


Figure 2-27 Sample Dialog activity

Note: A *Trigger* defines the event that causes an activity to start or to continue. Web activities have the following characteristics:

- ▶ Web activities always start with the e-Marketing Spot trigger.
By default, the e-Marketing Spot trigger is included as the first element in all standard Web activity templates. The Web activity is triggered when a customer views the page containing the e-Marketing Spot.
- ▶ A Web activity cannot have more than one trigger.

Dialog activities have the following characteristics:

- ▶ Dialog activities can start with any trigger that is available in the palette.
In most cases, triggers are customer events (for example, a customer registers or places an order). Additionally, a trigger can be an elapsed period of time (for example, a week). When the trigger that you specify occurs, the Dialog activity proceeds.
- ▶ A Dialog activity can have multiple triggers.

A Dialog activity supports the following specific triggers:

- ▶ Wait
- ▶ Customer Places Order
- ▶ Customer Abandons Shopping Cart
- ▶ Customer Registers
- ▶ Customer Participates in Social Commerce
- ▶ Customer is in Segment

A Dialog activity supports the following specific actions:

- ▶ Send Message
- ▶ Issue Coupon

2.4 Management Center enhancements

Management Center is the preferred tool for creating and managing promotions and marketing activities. In this section, we discuss the new features in Management Center with IBM WebSphere Commerce V7.

2.4.1 Activity templates

Activity templates provide a starting point to create a new marketing activity, avoiding any repetitive work. You can choose from the default templates provided by IBM WebSphere Commerce, or you can create new templates that suit your requirements. Activity templates and marketing activities share the same namespace. So, you must ensure that the name for an activity template is unique among all activity templates *and* marketing activities.

Figure 2-28 shows the standard templates that are available in Management Center.

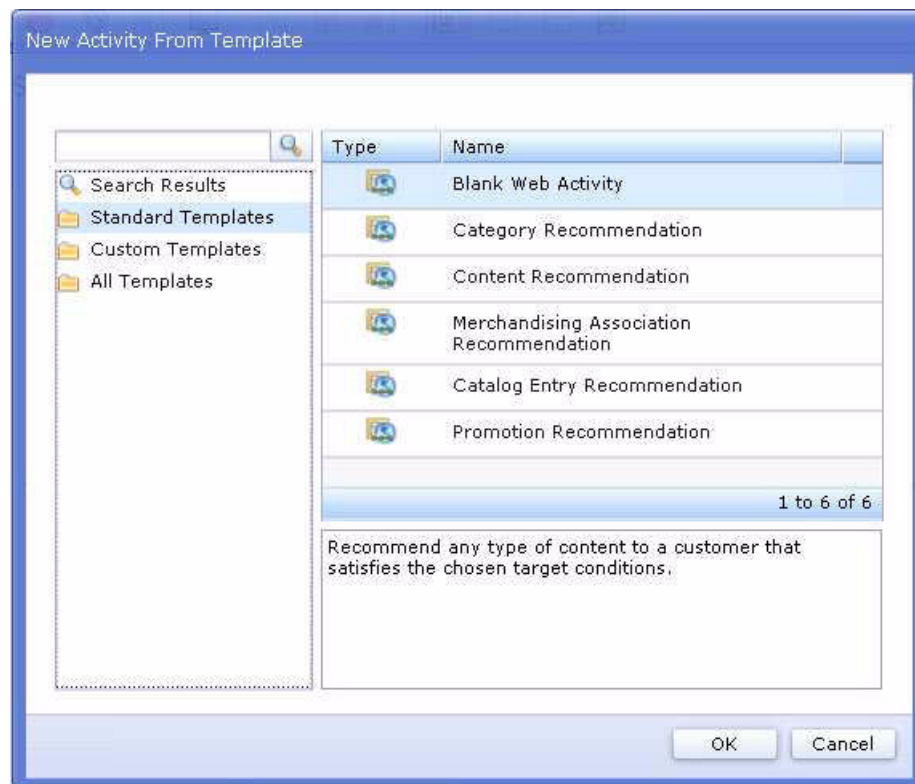


Figure 2-28 Activity template picker

2.4.2 Customer segments

Management Center now supports the ability to create customer segments. The UI is simple to use. A new option is added to allow marketing activities to add and remove customers automatically from a segment (Figure 2-29) so that you can support user segments implicitly through marketing activities without having to actively manage the same.

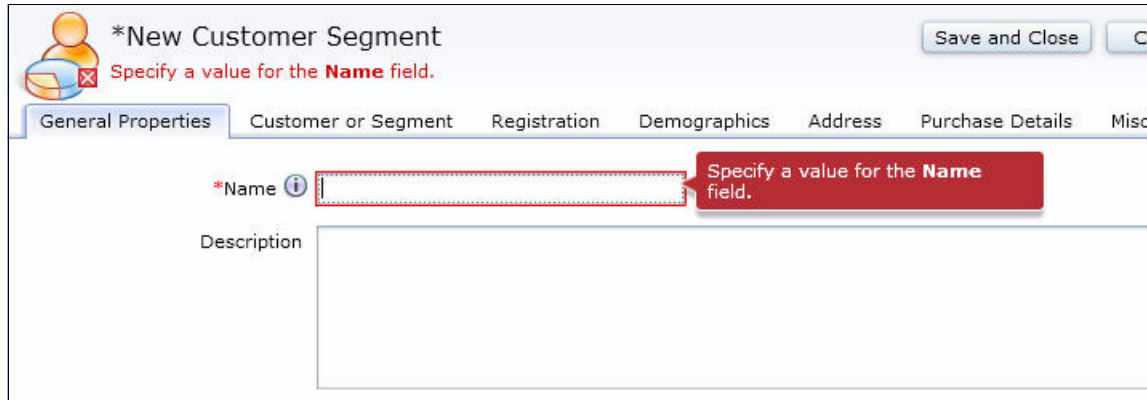


Figure 2-29 Create customer window in Management Center

2.4.3 Statistics

Management Center can display the statistics that the IBM WebSphere Commerce marketing engine collects. These statistics provide a good idea of the effectiveness of various Web activities. You can check the statistics at any time while the activity is running or after it becomes inactive.

Experiment statistics

These results help determine whether the current marketing effort is effectively maximizing revenue by targeting the right audience with the most appropriate message. If not, incorporating the experiment path into the control path might improve the efficiency of the marketing message. Figure 2-30 shows the experiment statistics view in Management Center.

Tip: Experiments in the Management Center support multiple experiment paths in any given experiment. The optional paths are called *experiment paths*, and the original path is called the *control path*.

General Properties Paths Statistics									
Number of Unique Customers		2							
Paths	* Name	* Winner	Views	View Orders	View Revenue	Clicks	Click Orders	Click Revenue	Currency
	bottom		4	0	0.00	0	0	0.00	
	top		2	0	0.00	0	0	0.00	
1 to 2 of 2									

Figure 2-30 Experiment statistics

Customer counters in campaign activities

The Activity Builder displays customer counters under each element in a campaign activity (Figure 2-31). The number indicates how many customers have reached a particular trigger, target, or action in the activity since it was activated. By default, the customer count numbers are updated every 15 minutes.

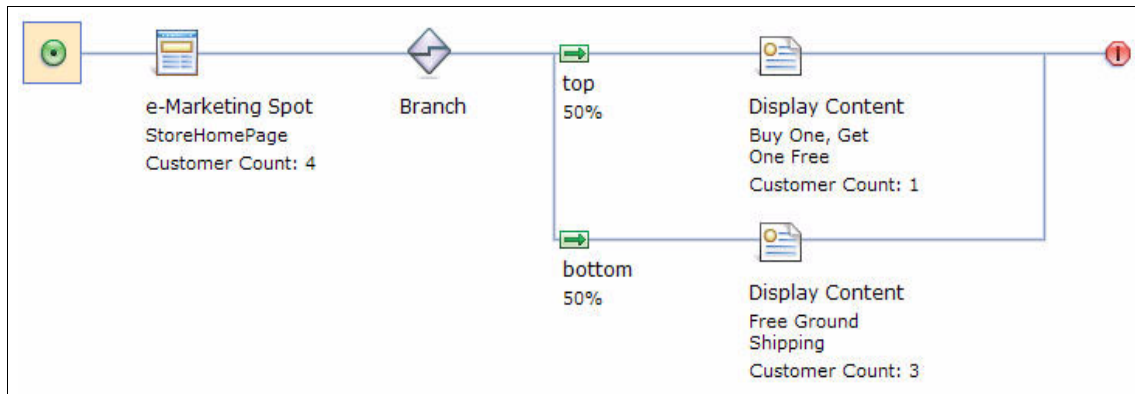


Figure 2-31 Activity element counters in the Web activity diagram

E-Marketing spot statistics

The Marketing tool displays statistics for e-Marketing Spots used in Web activities. For a given activity, the statistics show the number of times the server has displayed content in the e-Marketing Spot, the number of times a customer has clicked the content, and the associated click-through rate (Figure 2-32).

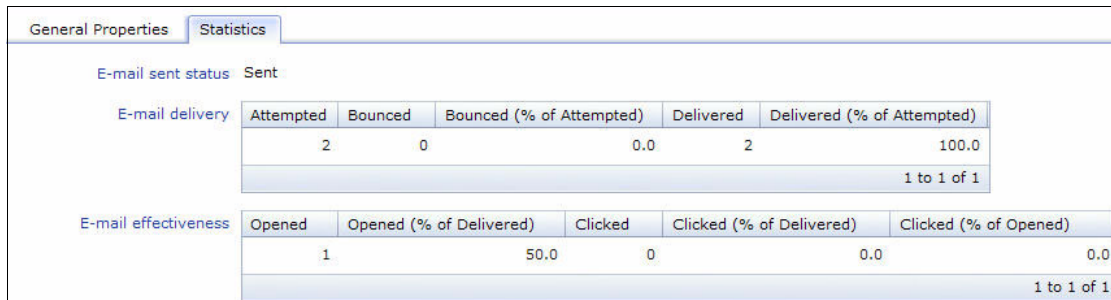


* Type	* Name	Views	Clicks	Click-through rate
	CategoryPageRecommendations	20	1	1 to 1 of 1

Figure 2-32 E-Marketing spot statistics

E-mail activity statistics

The Marketing tool displays statistics for e-mail activities. For a given e-mail activity, you can review statistics that are related to both e-mail delivery and e-mail effectiveness as shown in Figure 2-33. This data can help you evaluate the effectiveness of e-mail activities while they are in progress and when they are complete.



E-mail sent status: Sent				
E-mail delivery				
Attempted	Bounced	Bounced (% of Attempted)	Delivered	Delivered (% of Attempted)
2	0	0.0	2	100.0
1 to 1 of 1				

E-mail effectiveness				
Opened	Opened (% of Delivered)	Clicked	Clicked (% of Delivered)	Clicked (% of Opened)
1	50.0	0	0.0	0.0
1 to 1 of 1				

Figure 2-33 E-mail activity statistics

2.5 Utilities

IBM WebSphere Commerce V7 introduces a new data loading utility to allow you to upgrade from older editions and to use the latest features available in the new edition.

Data load refers to loading data from external data sources to WebSphere Commerce database. Typically, data loading includes the following scenarios:

- ▶ *Initial data loading* is the first time that you load the data into the database. Usually, a large amount of data is involved with the initial data loading.
- ▶ *Delta load* is used for data insert, update, and delete. Delta load can happen daily or weekly.

IBM WebSphere Commerce V7 improves upon the **massload** tool for data loading in earlier versions. Now, loading data is a single step data load from a CSV file to an IBM WebSphere Commerce DB. There is no need to generate intermediate files.

This new data load solution is based on the WebSphere Commerce business objects. To use this new data load tool, you need to understand the WebSphere Commerce business object schema instead of the physical database schema. You now need to have only CSV input files and to map the columns in the CSV file to the components of the logical business objects using XPath.

In IBM WebSphere Commerce V7, data load supports catalog and catalog related components, such as catalog group, catalog entry, merchandise association; as well as price and inventory for the catalog entries. Other data such as orders, promotions, and contracts will be delivered in future releases.

In IBM WebSphere Commerce V7, you can still use the **massload** tool. If you want to load components other than catalog, price, and inventory, you can either customize the new data load or use the existing **massload** tool.

2.6 Stack update

IBM WebSphere Commerce V7 builds upon the latest Java EE 5 technology and provides a state-of-the-art development tool using IBM Rational Application Developer V7.5. The following software is compatible for IBM WebSphere Commerce V7:

- ▶ IBM WebSphere Application Server V7
- ▶ IBM Rational Application Developer V7.5
- ▶ IBM DB2 V9.5

For a complete list of supported operating systems, databases, servers, and browsers, refer to the WebSphere Commerce Version 7 Information Center:

<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp>



Distributed order management integration

This chapter provides an overview of WebSphere Commerce order and inventory management systems and distributed order management (referred to as *DOM* in this book). It also provides information about the DOM integration framework as well as a sample DOM integration scenario with detailed steps.

This chapter includes the following topics:

- ▶ Overview of WebSphere Commerce order management and inventory management
- ▶ Integration of WebSphere Commerce with the DOM solution
- ▶ Implementation of WebSphere Message Broker mediation module for DOM integration
- ▶ Implementing WebSphere Enterprise Service Bus mediation module for DOM integration
- ▶ Configuring the DOM integration feature

3.1 Overview of WebSphere Commerce order management and inventory management

The WebSphere Commerce order management and inventory management consists of the following steps:

1. Orders are captured online by the storefront or through other channels, such as call center, kiosks, and so on.
2. The orders are sent to the WebSphere Commerce order management system for further processing.
3. During the order capturing and processing, inventory services such as checking inventory availability, reserving inventory, and reversing inventory reservation are needed. WebSphere Commerce informs the inventory management systems about available to promise (ATP) and non-ATP.

3.1.1 WebSphere Commerce order management

Order capture and processing are important components in an electronic commerce system. In WebSphere Commerce, there are different channels for order capture:

- ▶ Online shoppers can submit orders from the storefront.
- ▶ Shoppers can place a call to an IBM WebSphere Commerce call center to ask a customer service representative to place an order.
- ▶ Using a mobile phone, a shopper can purchase items by using the mobile storefront.
- ▶ A shopper can purchase items by using a kiosk.

Detailed steps about placing an order in WebSphere Commerce Madisons Starter Store can be found in 4.2, “Buy online, pick up in store” on page 134.

After the order is placed, an administrator can log on to WebSphere Commerce Accelerator to find the order and process it.

Figure 3-1 illustrates the user interface in WebSphere Commerce Accelerator that is used to find an order.

Select ▾ Jacksonville Outlet - Madisons - United States English

Store Marketing Merchandise Auctions Operations Payments Help

Logout > Home > Find Orders

Find Cancel

Find Orders

To search for a customer order, type information in one or more of the fields below and click **Find**. For a more refined search, click **Advanced search** below.

Order number

Customer logon ID
 Match case, beginning with ▾

[Advanced search](#)

Figure 3-1 Find orders in WebSphere Commerce Accelerator

After orders are found, the administrator can do future processing on them. For the orders that have ATP inventory or non-ATP inventory through WebSphere Commerce, the order flow after capture is as follows:

1. Approve the order, if needed.
2. Approve the payment of the order.
3. Release the orders to the fulfillment centers.
4. Create a pick batch.
5. View and print pick tickets.
6. View and print packaging slips.
7. Pick and pack the products.
8. Create a package.
9. Ship the goods to customers.
10. Confirm the shipment.
11. Close the order.

For more information, refer to the *Order flow process* topic in the WebSphere Commerce Version 7 Information Center at:

<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.user.doc/concepts/coosorderflowprocessgeneral.htm>

For DOM inventory, after the orders have been captured, they are transferred to the DOM system. The orders' status will be synchronized with DOM when needed. More operations to process the order are performed by the DOM system.

To better support the order capture, order management, and order integration with external system, WebSphere Commerce supplies order services through the service-oriented architecture (SOA) approach, which include:

- ▶ GetOrder service
- ▶ ChangeOrder service
- ▶ ProcessOrder service
- ▶ SyncOrder service

The GetOrder service is used to retrieve the order information for an order. It is usually used in a Web 2.0 starter store to display order information to online shoppers. For more information about the GetOrder service, refer to the *Get Order* topic in the WebSphere Commerce Version 7 Information Center at:

<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.webservices.doc/refs/rwvgetorder.htm>

The ChangeOrder service can be used to create a new order, add an order item to an order, and update the order. When online shoppers add something to their shopping cart or modify their shopping cart, this service will be invoked.

The ProcessOrder service is used to perform processes on the shoppers order. For example, when the online shoppers try to check out, the process service will be invoked to do some preparations for the order.

The SyncOrder service is used to synchronize the order information with external order management systems. Usually when the order in an external order management system has been changed, the external order management system can invoke this order service to synchronize the order status in WebSphere Commerce.

For more information about these order service, refer to the *SOI and BOD service modules* topic in the WebSphere Commerce Version 7 Information Center at:

<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.developer.soa.doc/concepts/csdcompare.htm>

Note: The order system of WebSphere Commerce can be integrated seamlessly into an SOA system.

3.1.2 WebSphere Commerce inventory management

WebSphere Commerce supplies two types of inventory management systems, which satisfy different customer requirements:

- ▶ ATP inventory management system
- ▶ Non-ATP inventory management system

ATP inventory

In an ATP system, the inventory that is available in stock (on hand) plus the inventory that is on order, not including inventory that is currently being processed, reserved, or allocated to back orders, can be processed when processing orders.

With ATP inventory, if there is a sufficient inventory on hand, the order can be fulfilled with a fulfillment status of allocated. Otherwise, if there is no sufficient inventory on hand, but there is sufficient inventory expected in the future, the order can be fulfilled with a fulfillment status of back ordered. Thus, both orders can be successfully placed.

Non-ATP inventory system

The non-ATP inventory is a simpler model than ATP inventory. It can only handle the inventory on hand. If there is no inventory on hand, the order cannot be placed successfully, and there are no sufficient inventory warning messages.

For more information about non-ATP inventory, refer to the *Non-ATP inventory information model* topic in the WebSphere Commerce Version 7 Information Center at:

http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.data.doc/concepts/cin_iminventoryasset3.htm

WebSphere Commerce Accelerator supplies friendly user interfaces for the administrators to manage the ATP inventory and non-ATP inventory, for example, on hand quantity adjustment, expected inventory adjustment of ATP, and so on. Figure 3-2 shows how to adjust the available quantity of a product.

The screenshot shows a web application interface with a top navigation bar containing links: Store, Marketing, Merchandise, Auctions, Operations, Payments, and Help. Below the navigation bar is a breadcrumb trail: Logout > Home > Find Inventory > Search Results > Adjust Quantity. Two buttons, 'OK' and 'Cancel', are positioned to the right of the breadcrumb trail. The main content area is titled 'Inventory' and displays 'Current inventory: 101'. Under the heading 'Change inventory', there are two radio button options: 'Increase inventory by the following value' and 'Decrease inventory by the following value'. Below these options is a text input field. At the bottom of the form, under the heading 'If this product is selected for an order', there are two checked checkboxes: 'Update the inventory for this product' and 'Verify that inventory is available for this product'.

Figure 3-2 Adjust available quantity of a product

Note: Many of management tasks can be completed in IBM Management Center for WebSphere Commerce. The Management Center for IBM WebSphere Commerce is the next generation business user tool for managing online business tasks, for example, catalog, marketing, and promotional tasks. It was introduced in IBM WebSphere Commerce V6 Feature Pack 3. Some of the management tasks are only available in WebSphere Commerce Accelerator.

For more information about the IBM Management Center of WebSphere Commerce, refer to the following resources:

- ▶ The WebSphere Commerce Version 7 Information Center at:
<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.management-center.doc/tasks/ttfgeneral.htm>
- ▶ *WebSphere Commerce Line-Of-Business Tooling Customization*, SG24-7619, located at:
<http://www.redbooks.ibm.com/redbooks/pdfs/sg247619.pdf>

Enhancement of ATP inventory and non-ATP inventory in IBM WebSphere Commerce V7

In IBM WebSphere Commerce V7, ATP inventory and non-ATP inventory have been enhanced to support the buy online and pickup in store scenarios (referred to as *BOPIS* in this book). For more detailed information about BOPIS, refer to 4.2, “Buy online, pick up in store” on page 134.

After the enhancement, the `GetInventoryAvailability` services can be used to retrieve the inventory availability of catalog entries in online stores or stores. This service is used by the product detail display page of the Madisons Starter Store to display the inventory availability to the online shoppers.

The online shoppers can complete an end-to-end BOPIS scenario with ATP and non-ATP inventory systems, just as they can do with the DOM integration solutions. (For more information about DOM, refer to 3.2, “Integration of WebSphere Commerce with the DOM solution” on page 84.) Shoppers can log on to an online store, select their favorite store from the Store Locator, add a product into the shopping cart, and pick it up at their favorite store.

For step-by-step guides of BOPIS scenarios, refer to 4.2.2, “BOPIS in Madisons Starter Store” on page 135.

3.2 Integration of WebSphere Commerce with the DOM solution

In this section, we discuss the WebSphere Commerce DOM solution, including an overview of the DOM solution, the integration structure of DOM, and the general steps to develop a mediation module of DOM.

3.2.1 WebSphere Commerce DOM solution overview

DOM is an external order management system or application responsible for processing the order, editing the order, and releasing the order to the appropriate fulfillment system. In most cases, inventory will be also be managed by this external system.

For an example scenario, a customer already has an existing order management system or wants to use a third-party order management system to process their online orders captured from WebSphere Commerce. The external order management system will be responsible for processing the order, editing the order, and releasing the order to the appropriate fulfillment system, as well as the orders returned.

WebSphere Commerce supplies new features to support this DOM system. As of Version 6 Feature Pack 5, DOM integration is part of the cross channel integration solutions in WebSphere Commerce. It is a back-end system integration that enables integration with DOM systems to provide a comprehensive coverage of the order life cycle, from capture to fulfillment across channels.

With DOM, the orders are captured from the WebSphere online storefront. These orders are transferred to external order management systems by WebSphere Commerce order Web services. The orders status are changed in external order management systems and synchronized with WebSphere Commerce for management purposes.

DOM integration supplies the following new features:

- ▶ A new inventory component with SOA services to check inventory availability and process inventory requirements, including caching and DOM integration capabilities

There is great flexibility in the inventory cache, which can use a distributed object cache mechanism in WebSphere Application Server or WebSphere Commerce databases through different configurations.

In the WebSphere Application Server distributed object cache mechanism, the DOM inventories are cached in the memory. The memory cache can serve the inventory request with a quicker response. In a cluster environment, the inventory caches are synchronized between different cluster members.

For more information about IBM WebSphere Application Server distributed object caches, refer to the following topics in the WebSphere Commerce Version 7 Information Center:

- *Using the DistributedMap and DistributedObjectCache interfaces for the dynamic cache*
http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp?topic=/com.ibm.websphere.nd.doc/info/ae/ae/tdyn_distmap.html
- *Object cache instance settings*
http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/topic/com.ibm.websphere.nd.doc/info/ae/ae/udyn_cacheinstancescollection.html
- *Java document for class DistributedObjectCache*
http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/topic/com.ibm.websphere.javadoc.doc/public_html/api/com/ibm/websphere/cache/DistributedObjectCache.html

There is an extended version of the dynamic cache monitor, which can help monitor the distributed object caches. For more information about how to configure and use this extended dynamic cache monitor, refer to the following address:

http://www.ibm.com/developerworks/websphere/downloads/cache_monitor.html

With the WebSphere Commerce database cache, the DOM inventory records are resident in DB2 databases or Oracle databases. WebSphere Commerce has supplied an efficient approach to access these databases by way of a data service layer (DSL).

For more information about WebSphere Commerce DSL, refer to the *Data service layer* topic in the WebSphere Commerce Version 7 Information Center at:

<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/topic/com.ibm.commerce.developer.soa.doc/concepts/csddsl.htm?resultof=%22%44%73%6c%22%20%22%64%73%6c%22%20>

Whether the local DOM inventory cache is in memory or in database, it is configured in the WebSphere Commerce database. Store administrators can change these configurations to change the cache locations.

- ▶ A new store component with services for store location

This new component gives WebSphere Commerce online shoppers the capability to find nearby stores based on entered locations. The shoppers can then pick up merchandise after they shop online or can go to the stores directly for shopping.

- ▶ Enhancement of catalog browsing

The enhancement of catalog browsing flow enables the shoppers to view product availability for certain stores. Before a shopper can see the availability, shoppers need to use Store location feature to select their favorite stores. They can also select the their favorite stores in the product detailed information display page.

- ▶ Buy online, pick up in store support (referred to as *BOPIS* in this book)

The online shoppers can buy online and pick up the merchandise in their favorite store.

As of WebSphere Commerce V7, the BOPIS scenarios are supported with the ATP and non-ATP inventory systems.

For more information about BOPIS, refer to 4.2, “Buy online, pick up in store” on page 134.

3.2.2 Shopping flow with DOM

Integrated with DOM system, the shopping flow (which exploits the benefits of DOM system) of the WebSphere Commerce Madisons Starter Store is composed of several steps, as shown in Figure 3-3.

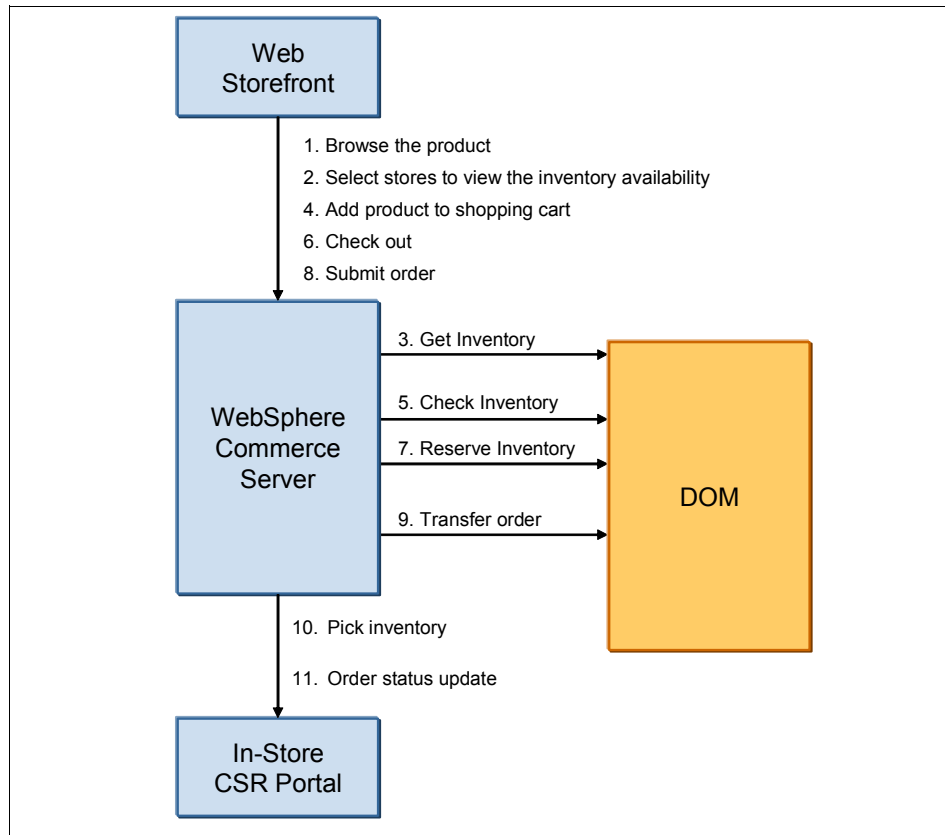


Figure 3-3 Shopping flow in DOM integration solution

The shopping flow uses the following steps:

1. The online shopper finds a product.
2. The online shopper selects their favorite store to check out.
3. WebSphere Commerce interacts with the DOM system to obtain the inventory availability of the product in the shopper's favorite store.
4. The shopper adds the product to the shopping cart.
5. WebSphere Commerce interacts with DOM to check whether there is sufficient inventory.

6. The shopper checks out.
7. WebSphere Commerce interacts with DOM to reserve inventory for this order.
8. The online shopper submits the order.
9. WebSphere Commerce transfers the order to DOM.
10. The online shopper goes to the store and picks up the product.
11. The order status is updated.

3.2.3 DOM integration structure

The WebSphere Commerce runtime component supplies a flexible framework to integrate back-end systems. The DOM integration solution works elegantly within this integration framework.

Figure 3-4 shows the overall structure of the DOM integration solution.

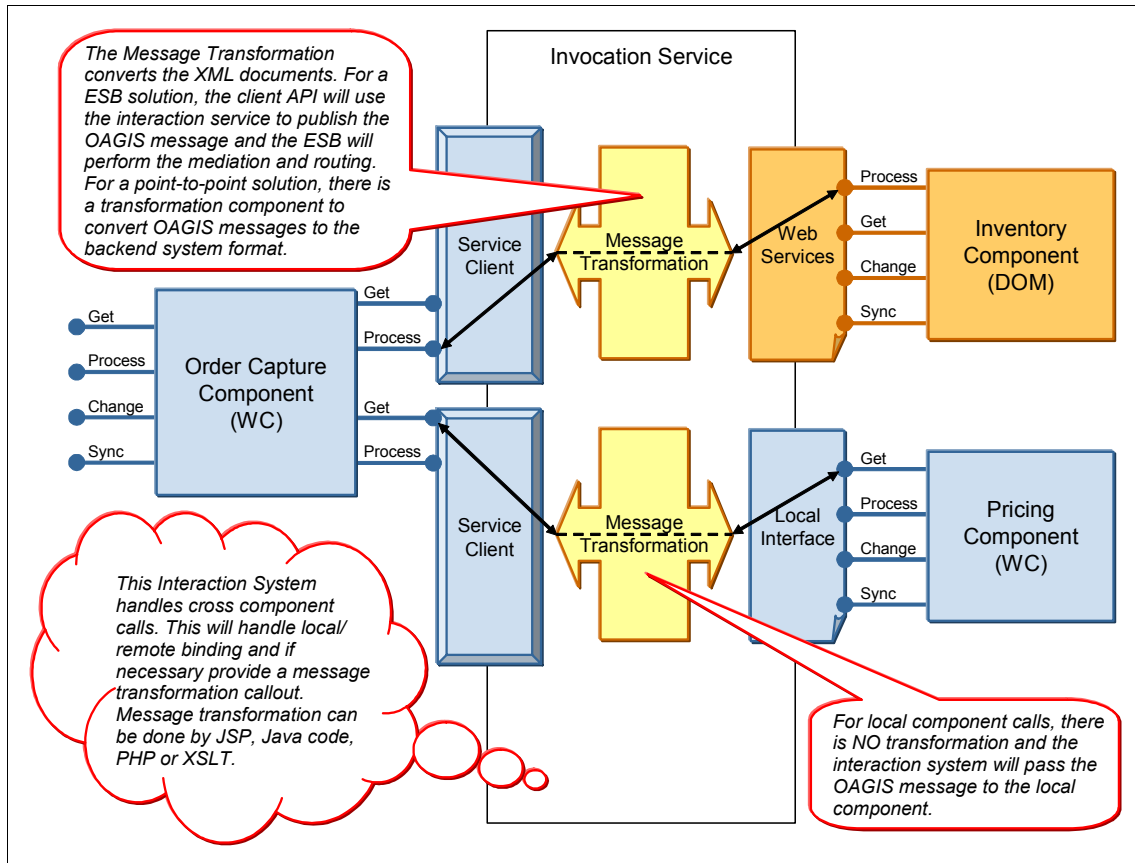


Figure 3-4 DOM integration structure

The order capture component is supplied by WebSphere Commerce, which is invoked by the online storefront to capture the orders. WebSphere Commerce supplies different order services to order processing, including the GetOrder service, the ProcessOrder service, the ChangeOrder service, and the SyncOrder service. These services enable the storefront to access the order in a flexible and loosely coupled approach.

The WebSphere Commerce SOA service client is the asset that is supplied by WebSphere Commerce. It can be used to invoke the SOA services through different bindings. On the WebSphere Commerce side, the service client can be used to invoke the external services or invoke services from different components inside of WebSphere Commerce.

The service client supplies utilities to compose the service message and to invoke the service, which can be used in the DOM side to gain quick access to WebSphere Commerce service.

The outgoing service messages from the client APIs are taken as WebSphere Commerce outbound service messages.

The message transformation module works to transform the WebSphere Commerce service messages to DOM messages and vice versa. For example, the request to the GetInventoryAvailability service will be transferred to format that can be recognized by the DOM inventory system. When the response from DOM inventory comes, the message transformation module transfers the response message to a format that can be understood by WebSphere Commerce. This transformation module can be developed and deployed in IBM WebSphere Message Broker or IBM WebSphere Enterprise Service Bus.

Note: For more information about how to development and deploy mediation modules, refer to 3.3, “Implementation of WebSphere Message Broker mediation module for DOM integration” on page 98, and 3.4, “Implementing WebSphere Enterprise Service Bus mediation module for DOM integration” on page 126.

The DOM inventory management module is the system in which inventory is managed and is the back-end system to WebSphere Commerce. DOM inventory can expose inventory services such as inventory query, inventory reservation, and inventory cancellation with its own message format. It can only communicate with WebSphere Commerce with the help of a message transformation module.

Note: When WebSphere Commerce acts as a service consumer, a component client API is called from the WebSphere Commerce task command. The client API uses the invocation service, and the invocation service requires a deployed configuration file to determine how to communicate with the remote component. Each component has a separate configuration file to configure the client API. Each store can also have a version of the configuration file that takes precedence over the default configuration, which allows the store to override some of or the entire configuration without changing the default configuration.

3.2.4 DOM integration outbound messages

A business object document (BOD) is a representation of a standard business process that flows within an organization or between organizations. BODs are defined by the Open Applications Group (OAG) using XML.

The WebSphere Commerce SOA service messages comply with this BOD standard. There is an application area and a data area in WebSphere Commerce service message. There is a noun part and a verb part in the data area, which indicates that a verb takes action on the noun. For more information about WebSphere Commerce SOA and BOD, refer to the following topics in the WebSphere Commerce Version 7 Information Center:

- ▶ *SOI and BOD service modules*

<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.developer.soa.doc/concepts/csdcompare.htm>

- ▶ *Business Object Document (BOD)*

http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.base.doc/misc/Business_Object_Document_%28BOD%29.htm

The DOM integration feature includes the following WebSphere Commerce outbound messages:

- ▶ GetInventoryAvailability/ShowInventoryAvailability
- ▶ ProcessInventoryRequirement/AcknowledgeInventoryRequirement with action code ReserveInventory
- ▶ ProcessInventoryRequirement/AcknowledgeInventoryRequirement with action code CancelInventoryReservation
- ▶ ProcessOrder/AcknowledgeOrder with action code TransferOrder

These outbound messages are transformed by the message transform module so that DOM can recognize that these messages can communicate with WebSphere Commerce.

The GetInventoryAvailability message is used to retrieve the inventory availability of catalog entries in online stores or in physical stores. The response to the request is in the ShowInventoryAvailability message.

The ProcessInventoryRequirement message processes the orders. Different action codes can be used in the request message for different purposes:

- ▶ CheckInventory
The action code CheckInventory checks the inventory availability for the order items in an order.
- ▶ ReserveInventory
The action code ReserveInventory reserves inventory from the DOM inventory system.
- ▶ DecrementCache
The action code DecrementCache decrements the local DOM inventory cache when inventories are reserved successfully from the DOM inventory.
- ▶ CancelInventoryReservation
The action code CancelInventoryReservation cancels an inventory reservation from the DOM inventory.

Among these action codes, ReserveInventory and CancelInventoryReservation are used as the outbound service messages, which are transformed before going to DOM. Other action codes are used internally by WebSphere Commerce in the DOM solution.

The ProcessOrder message with action code TransferOrder is used to transfer the orders captured by the WebSphere Commerce Madisons Starter Store to DOM. After a successful transfer, the orders have life cycles in DOM.

The sample message in Example 3-1 retrieves the inventory availability of catalog entry ID 10002 in online stores 10001, 10037, and 10026.

Example 3-1 GetInventoryAvailability request sample message

```
<_inv:GetInventoryAvailability releaseID="9.0" versionID="7.0.0.0"
xmlns:_inv="http://www.ibm.com/xmlns/prod/commerce/9/inventory"
xmlns:_wcf="http://www.ibm.com/xmlns/prod/commerce/9/foundation"
xmlns:oa="http://www.openapplications.org/oagis/9">
  <oa:ApplicationArea xsi:type="_wcf:ApplicationAreaType">
    <oa:CreationDateTime>2009-08-10T18:23:32.796Z</oa:CreationDateTime>
    <oa:BODID>e8ec0441-85da-11de-aea5-84214a8064d5</oa:BODID>
    <_wcf:BusinessContext/>
  </oa:ApplicationArea>
  <_inv:DataArea>
    <oa:Get>
      <oa:Expression
expressionLanguage="_wcf:XPath">{_wcf.ap=IBM_Store_Details}/InventoryAv
ailability[InventoryAvailabilityIdentifier/ExternalIdentifier[CatalogEn
```

```

tryIdentifier[(UniqueID='10002')] and
(OnlineStoreIdentifier[(UniqueID='10001')] or
PhysicalStoreIdentifier[(UniqueID='10037' or
UniqueID='10026')]))]]</oa:Expression>
</oa:Get>
</_inv:DataArea>
</_inv:GetInventoryAvailability>

```

Note: There is one verb get for the data area of the GetInventoryAvailability BOD. There is no noun for this message, because all the necessary information to retrieve inventory availability is contained in the expression.

In WebSphere Commerce, each expression has several parameters. In this sample:

- ▶ The expression `expressionLanguage="_wcf:XPath"` specifies that the expression should be parsed by XPath.
- ▶ The expression `_wcf.ap=IBM_Store_Details` specifies that this request uses an access profile of IBM_Store_Details.

The remaining part of the expression is the XPath to retrieve inventory availability, which is used by DSL to retrieve the data from databases.

You can find more information in the *Get Request and the Show Response* topic in the WebSphere Commerce Version 7 Information Center:

<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.webservices.doc/concepts/cwvget.htm>

The sample response in Example 3-2 gives a confirmation to the request that there is sufficient inventories with 555 available for online store 10001, 200 available for store 10037, and 959 available for store 10026.

Example 3-2 ShowInventoryAvailability response sample

```

<_inv:ShowInventoryAvailability releaseID="9.0"
xmlns:_inv="http://www.ibm.com/xmlns/prod/commerce/9/inventory"
xmlns:_wcf="http://www.ibm.com/xmlns/prod/commerce/9/foundation"
xmlns:oa="http://www.openapplications.org/oagis/9">
  <oa:ApplicationArea xsi:type="_wcf:ApplicationAreaType">
    <oa:CreationDateTime>2009-08-10T18:23:34.921Z</oa:CreationDateTime>
    <oa:BODID>ea304410-85da-11de-aea5-84214a8064d5</oa:BODID>
  </oa:ApplicationArea>
  <_inv:DataArea>
    <oa:Show recordSetCompleteIndicator="true" recordSetCount="3"
recordSetStartNumber="0" recordSetTotal="3">

```

```

    <oa:OriginalApplicationArea>
      <oa:CreationDateTime>2009-08-10Z</oa:CreationDateTime>
      <oa:BODID>e8ec0441-85da-11de-aea5-84214a8064d5</oa:BODID>
    </oa:OriginalApplicationArea>
  </oa:Show>
  <_inv:InventoryAvailability>
    <_inv:InventoryAvailabilityIdentifier>
      <_wcf:ExternalIdentifier>
        <_wcf:CatalogEntryIdentifier>
          <_wcf:UniqueID>10002</_wcf:UniqueID>
          <_wcf:ExternalIdentifier>
            <_wcf:PartNumber>FULO-0101</_wcf:PartNumber>
          </_wcf:ExternalIdentifier>
        </_wcf:CatalogEntryIdentifier>
        <_wcf:OnlineStoreIdentifier>
          <_wcf:UniqueID>10001</_wcf:UniqueID>
          <_wcf:ExternalIdentifier>
            <_wcf:NameIdentifier>Madisons</_wcf:NameIdentifier>
          </_wcf:ExternalIdentifier>
        </_wcf:OnlineStoreIdentifier>
      </_wcf:ExternalIdentifier>
    </_inv:InventoryAvailabilityIdentifier>
    <_inv:InventoryStatus>Available</_inv:InventoryStatus>
    <_inv:AvailableQuantity uom="C62">555.0</_inv:AvailableQuantity>
  </_inv:InventoryAvailability>
  <_inv:InventoryAvailability>
    <_inv:InventoryAvailabilityIdentifier>
      <_wcf:ExternalIdentifier>
        <_wcf:CatalogEntryIdentifier>
          <_wcf:UniqueID>10002</_wcf:UniqueID>
          <_wcf:ExternalIdentifier>
            <_wcf:PartNumber>FULO-0101</_wcf:PartNumber>
          </_wcf:ExternalIdentifier>
        </_wcf:CatalogEntryIdentifier>
        <_wcf:PhysicalStoreIdentifier>
          <_wcf:UniqueID>10037</_wcf:UniqueID>
          <_wcf:ExternalIdentifier>Calgary Circle
Mall</_wcf:ExternalIdentifier>
        </_wcf:PhysicalStoreIdentifier>
      </_wcf:ExternalIdentifier>
    </_inv:InventoryAvailabilityIdentifier>
    <_inv:InventoryStatus>Available</_inv:InventoryStatus>
    <_inv:AvailableQuantity uom="C62">200.0</_inv:AvailableQuantity>
  </_inv:InventoryAvailability>
  <_inv:InventoryAvailability>

```

```

    <_inv:InventoryAvailabilityIdentifier>
      <_wcf:ExternalIdentifier>
        <_wcf:CatalogEntryIdentifier>
          <_wcf:UniqueID>10002</_wcf:UniqueID>
          <_wcf:ExternalIdentifier>
            <_wcf:PartNumber>FULO-0101</_wcf:PartNumber>
          </_wcf:ExternalIdentifier>
        </_wcf:CatalogEntryIdentifier>
        <_wcf:PhysicalStoreIdentifier>
          <_wcf:UniqueID>10026</_wcf:UniqueID>
          <_wcf:ExternalIdentifier>Calgary
Mall</_wcf:ExternalIdentifier>
        </_wcf:PhysicalStoreIdentifier>
      </_wcf:ExternalIdentifier>
    </_inv:InventoryAvailabilityIdentifier>
    <_inv:InventoryStatus>Available</_inv:InventoryStatus>
    <_inv:AvailableQuantity uom="C62">959.0</_inv:AvailableQuantity>
  </_inv:InventoryAvailability>
</_inv:DataArea>
</_inv:ShowInventoryAvailability>

```

Note: The data area of ShowInventoryAvailability BOD is composed of one verb show and three InventoryAvailability nouns. Each noun is for a combination of a catalog entry and a store.

3.2.5 DOM integration inbound message

A list of inbound messages is used for the DOM integration solution:

- ▶ SyncInventoryAvailability/ConfirmBOD
- ▶ SyncOrder/ConfirmBOD

SyncInventoryAvailability/ConfirmBOD is used by the SyncInventoryAvailability service to synchronize the inventory availability information from DOM to WebSphere Commerce. This service can be invoked by DOM directly, which does not need a message transformation.

Upon every SyncInventoryAvailability service messages received, WebSphere Commerce updates its local DOM inventory cache. The distributed object memory cache or the database cache will be updated based on the configuration.

SyncOrder/ConfirmBOD is used by the SyncOrder service to synchronize the order information from DOM to WebSphere Commerce. The DOM system can

invoke this service directly through the WebSphere Commerce SOA service client. Thus, transformation of messages is not needed. Every time that the order status is changed in the DOM system we recommend that you call this service to synchronize the updated order status to WebSphere Commerce.

In Example 3-3, the DOM system used this message to synchronize the inventory availability of catalog entry ID 50400000173 and store ID 555. After this message, the local inventory cache of the inventory availability is updated to the available quantity of 135.0 and the available status.

Example 3-3 SyncInventoryAvailability sample request message

```
<_inv:SyncInventoryAvailability releaseID="9.0" versionID="7.0.0.0"
xmlns:_inv="http://www.ibm.com/xmlns/prod/commerce/9/inventory"
xmlns:_wcf="http://www.ibm.com/xmlns/prod/commerce/9/foundation"
xmlns:oa="http://www.openapplications.org/oagis/9"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <oa:ApplicationArea xsi:type="_wcf:ApplicationAreaType">
    <oa:CreationDateTime>2009-08-12T07:16:54.218Z</oa:CreationDateTime>
    <oa:BODID>1ccd80f0-8710-11de-9a77-841b4a826c66</oa:BODID>
    <_wcf:BusinessContext>
      <_wcf:ContextData name="langId">-1</_wcf:ContextData>
      <_wcf:ContextData name="storeId">555</_wcf:ContextData>
    </_wcf:BusinessContext>
  </oa:ApplicationArea>
  <_inv:DataArea>
    <oa:Sync>
      <oa:ActionCriteria>
        <oa:ActionExpression actionCode="Change"
expressionLanguage="_wcf:XPath">/InventoryAvailability[1]</oa:ActionExp
ression>
      </oa:ActionCriteria>
    </oa:Sync>
  <_inv:InventoryAvailability>
    <_inv:InventoryAvailabilityIdentifier>
      <_wcf:ExternalIdentifier>
        <_wcf:CatalogEntryIdentifier>
          <_wcf:UniqueID>50400000173</_wcf:UniqueID>
        </_wcf:CatalogEntryIdentifier>
        <_wcf:OnlineStoreIdentifier>
          <_wcf:UniqueID>555</_wcf:UniqueID>
        </_wcf:OnlineStoreIdentifier>
      </_wcf:ExternalIdentifier>
    </_inv:InventoryAvailabilityIdentifier>
    <_inv:InventoryStatus>Available</_inv:InventoryStatus>
    <_inv:AvailableQuantity uom="C62">135.0</_inv:AvailableQuantity>
```

```

<_inv:AvailabilityDateTime>2008-08-08T13:36:15.64Z</_inv:AvailabilityDa
teTime>
  <_inv:AvailabilityOffset>1000234</_inv:AvailabilityOffset>
  <_wcf:UserData>
    <_wcf:UserDataField name="customField1">8</_wcf:UserDataField>
    <_wcf:UserDataField name="customField2">88</_wcf:UserDataField>
    <_wcf:UserDataField name="customField3">8-online store
field3</_wcf:UserDataField>
  </_wcf:UserData>
</_inv:InventoryAvailability>

```

Note: There is one verb Sync and one noun InventoryAvailability in this BOD. Usually, there are different action codes for the verb. With these different action codes, a verb can take different actions on a noun. In this sample, Sync the InventoryAvailability with action code Change will try to update the InventoryAvailability in the local inventory cache.

Example 3-4 is the confirmation message to DOM. The WebSphere Commerce side uses this message to inform DOM that the synchronization is complete.

Example 3-4 Sample response message to SyncInventoryAvailability

```

<oa:ConfirmBOD releaseID="9.0"
xmlns:_wcf="http://www.ibm.com/xmlns/prod/commerce/9/foundation"
xmlns:oa="http://www.openapplications.org/oagis/9"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <oa:ApplicationArea xsi:type="_wcf:ApplicationAreaType">
    <oa:CreationDateTime>2009-08-12T07:17:04.468Z</oa:CreationDateTime>
    <oa:BODID>22e9aea0-8710-11de-b60c-82804a812414</oa:BODID>
  </oa:ApplicationArea>
  <oa:DataArea>
    <oa:Confirm>
      <oa:OriginalApplicationArea>
        <oa:CreationDateTime>2009-08-12Z</oa:CreationDateTime>
        <oa:BODID>1ccd80f0-8710-11de-9a77-841b4a826c66</oa:BODID>
      </oa:OriginalApplicationArea>
    </oa:Confirm>
  </oa:DataArea>
  <oa:BOD>
    <oa:OriginalApplicationArea xsi:type="_wcf:ApplicationAreaType">
      <oa:CreationDateTime>2009-08-12Z</oa:CreationDateTime>
      <oa:BODID>1ccd80f0-8710-11de-9a77-841b4a826c66</oa:BODID>
      <_wcf:BusinessContext>
        <_wcf:ContextData name="langId">-1</_wcf:ContextData>
        <_wcf:ContextData name="storeId">555</_wcf:ContextData>
      </_wcf:BusinessContext>
    </oa:OriginalApplicationArea>
  </oa:BOD>
</oa:ConfirmBOD>

```

```
</_wcf:BusinessContext>
</oa:OriginalApplicationArea>
<oa:BODSuccessMessage/>
</oa:BOD>
</oa:DataArea>
</oa:ConfirmBOD>
```

Note: This simple ConfirmBOD message provides a confirmation message to the services consumer to indicate that the operation has completed successfully.

3.2.6 Integration steps

Generally, integrating the DOM integration solution requires the following steps:

1. Expose services or APIs to the DOM external system, for example WebSphere Commerce. In 3.3, “Implementation of WebSphere Message Broker mediation module for DOM integration” on page 98, we use a simple Web service as the DOM simulator. The DOM simulator supplies simple Web services for order management and inventory management with its own format.
2. Develop a message mediation module to do the transformation work between WebSphere Commerce messages and the DOM messages.
3. Deploy the message mediation module in an enterprise service bus, for example, IBM WebSphere Message Broker or IBM WebSphere Enterprise Service Bus.
4. Make configurations in WebSphere Commerce to communicate with the DOM system through the message mediation modules.

For more detailed steps, refer to 3.3, “Implementation of WebSphere Message Broker mediation module for DOM integration” on page 98.

3.3 Implementation of WebSphere Message Broker mediation module for DOM integration

For this exercise, we use an external OMS simulator, which is a simple JEE application written to serve as a sample back-end system for DOM Integration purposes. It exposes a number of inventory and order operations as Web services. It does not perform any real processing when generating a response. It

instead returns standard or random results, depending on the case, to use as sample sets of data.

You can find the EAR for the external OMS simulator and the installation steps in the *Installing the external OMS simulator* topic in the WebSphere Commerce Version 7 Information Center.

3.3.1 Installation


We must install WebSphere Message Broker toolkit, WebSphere Message Broker run time, WebSphere MQ server, and DB2 Universal Database. For installation of these products, refer to the product documentation and manuals.

3.3.2 Post-installation tasks

After a full installation of WebSphere Message Broker (both the toolkit and the run time), ensure that WebSphere MQ and DB2 are running using the steps that we describe in this section.


WebSphere MQ service

To verify that the WebSphere MQ service is running, perform one of the following tasks:

- ▶ Check the MQ system tray icon () to see whether WebSphere MQ is running. The green up arrow shows that the system tray icon for WebSphere MQ is in the started state. If WebSphere MQ is not running, right-click the system tray icon, and then click **Start WebSphere MQ**.
- ▶ Check the Services list for the IBM MQSeries® status. The status should be *Started*. To open Services, click **Start** ∅ **Control Panel** ∅ **Administrative Tools** ∅ **Services**. If WebSphere MQ is not running in the Services window, right-click **IBM MQSeries**, and then click **Start**.

DB2 Universal Database

To verify that DB2 Universal Database is running, perform one of the following tasks:

- ▶ Check the system tray icon () to see whether DB2 Universal Database is running. The green color indicates that the system tray icon for DB2 Universal Database is in the started state. If DB2 Universal Database is not running, right-click the system tray icon, and then click **Start** (DB2).
- ▶ At a command prompt type **db2start**. If DB2 Universal Database is running, the following message displays:

The database manager is already active

If DB2 Universal Database is not running, issuing the **db2start** command to start DB2 Universal Database.

- ▶ Check Services for the status of the DB2 - DB2-0 service, which should be *Started*. To open Services, click **Start** ∅ **Control Panel** ∅ **Administrative Tools** ∅ **Services**. If DB2 is not running (no status is displayed), in the Services window, right-click **DB2 - DB2-0**, and then click **Start**.

3.3.3 Create default configuration for WebSphere Message Broker


To run any application on WebSphere Message Broker, it must configure the runtime components and create a broker domain. The WebSphere Message Broker default configuration wizard creates the following components and resources to provide a simple broker domain for testing the applications:

- ▶ A Configuration Manager called WBRK6_DEFAULT_CONFIGURATION_MANAGER
- ▶ A broker called WBRK6_DEFAULT_BROKER
- ▶ A queue manager called WBRK6_DEFAULT_QUEUE_MANAGER, shared by the Configuration Manager and the broker
- ▶ A listener on the queue manager on port 2414
- ▶ A broker database called DEFBKDB6

Note: An explanation and discussion of these components of WebSphere Message Broker is out of scope of this book. For more information, refer to *WebSphere Message Broker Basics*, SG24-7137.

Steps to run the default configuration wizard

To run the default configuration wizard:

1. Open the WebSphere Message Broker Toolkit. If the WebSphere Message Broker Welcome window does not display, click **Help**  **Welcome**.
2. Click the Get Started icon shown in Figure 3-5. The Getting Started page of the Welcome window opens.

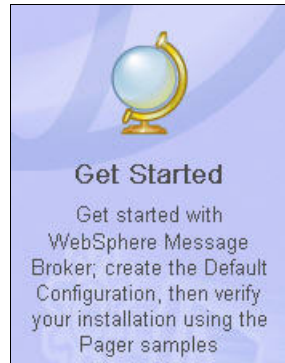


Figure 3-5 Get Started

3. Click the Create the Default Configuration icon (Figure 3-6). The Create the Default Configuration topic in the product documentation opens in the WebSphere Message Broker Toolkit help system.

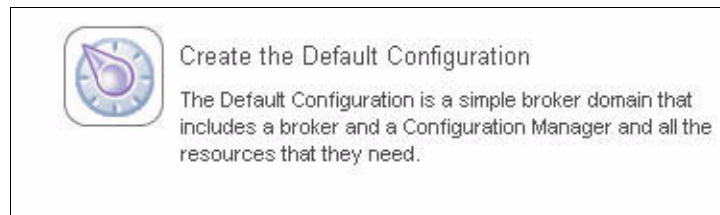


Figure 3-6 Create the Default Configuration

4. Click the **Start the Default Configuration** wizard link to start the wizard.

When the Default Configuration wizard has completed successfully, a simple broker domain is configured on the system. The connection to the broker domain is displayed in the Broker Administration perspective.

3.3.4 Creating mediation module in WebSphere Message Broker

This section explains the implementation of the mediation module for the ReserveInventory Outbound message (refer to 3.2.4, “DOM integration outbound messages” on page 91).

Creating new mediation module project

To create a new mediation module project named SampleDOMMediation:

1. In the WebSphere Message Broker toolkit switch to the Broker Application Development Perspective.

Note: Verify that Build Automatically is turned on for the WebSphere Message Broker toolkit by navigating to the Project menu on the top menu bar.

2. Click **Start from WSDL and/or XSD files** (Figure 3-7).

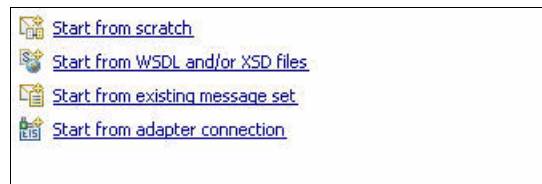


Figure 3-7 Launch

3. The New Message Broker Application window opens. Enter the values shown in Table 3-1 and in Figure 3-8, and click **Next**.

Table 3-1 Mediation module properties

Property	Value
Message flow project name	SampleDOMMediation
Message set project name	SampleDOMMediationMessageSet
Message set name	SampleDOMMediationMessageSet
Message flow name	SampleDOMMediationFlow
Working set name	SampleDOMMediation

Quick Start ✕

New Message Broker Application

Set up the basic resources required to develop a Message Broker application using WSDL and XSD files as a starting point.

Message flow project name:

Message set project name:

Message set name:

Message flow creation

☒ Create a new message flow in my flow project

Message flow name:

Working set creation

☒ Create a new working set for these resources

Working set name:

? < Back Next > Finish Cancel

Figure 3-8 New Message Broker Application

4. On the Resource selection screen, select the “Use external resources” option. Browse to the location of WSDL. For our example, we import the following WSDL:

InventoryServices.wsdl located in <WCDE install dir>/workspace/WebServicesRouter/WebContent/component-services/wsdl

Refer to Figure 3-9 for details.

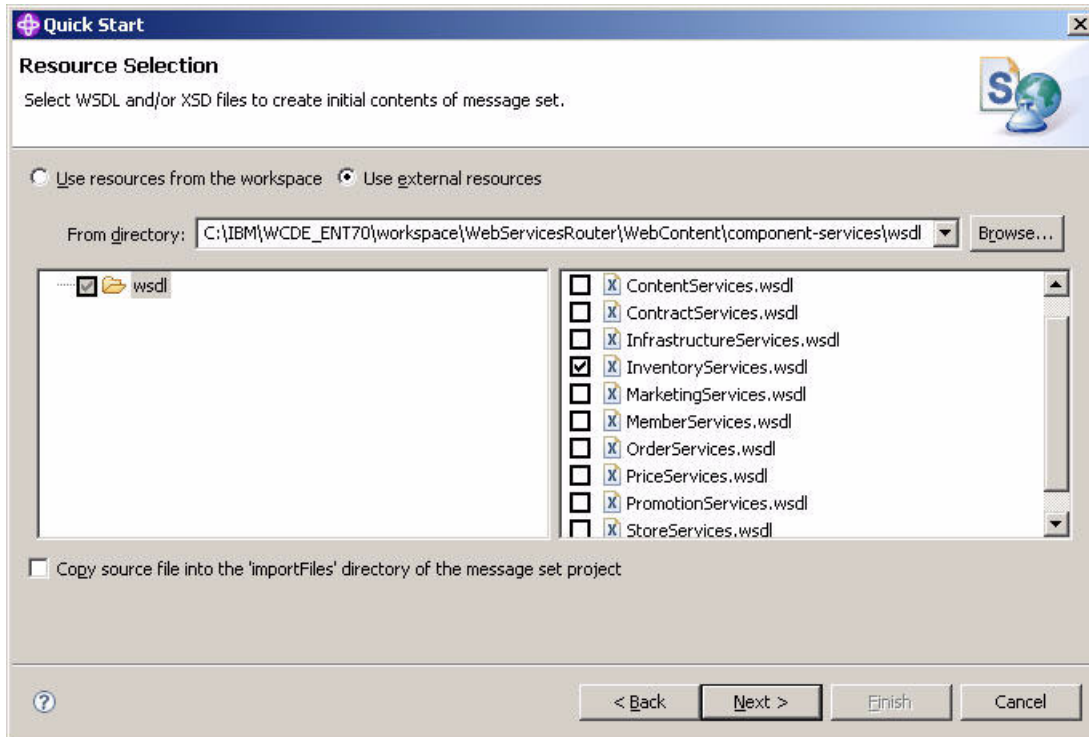


Figure 3-9 Resource Selection

5. Click **Next** to get the Binding Selection window (Figure 3-10). Click **Finish**.

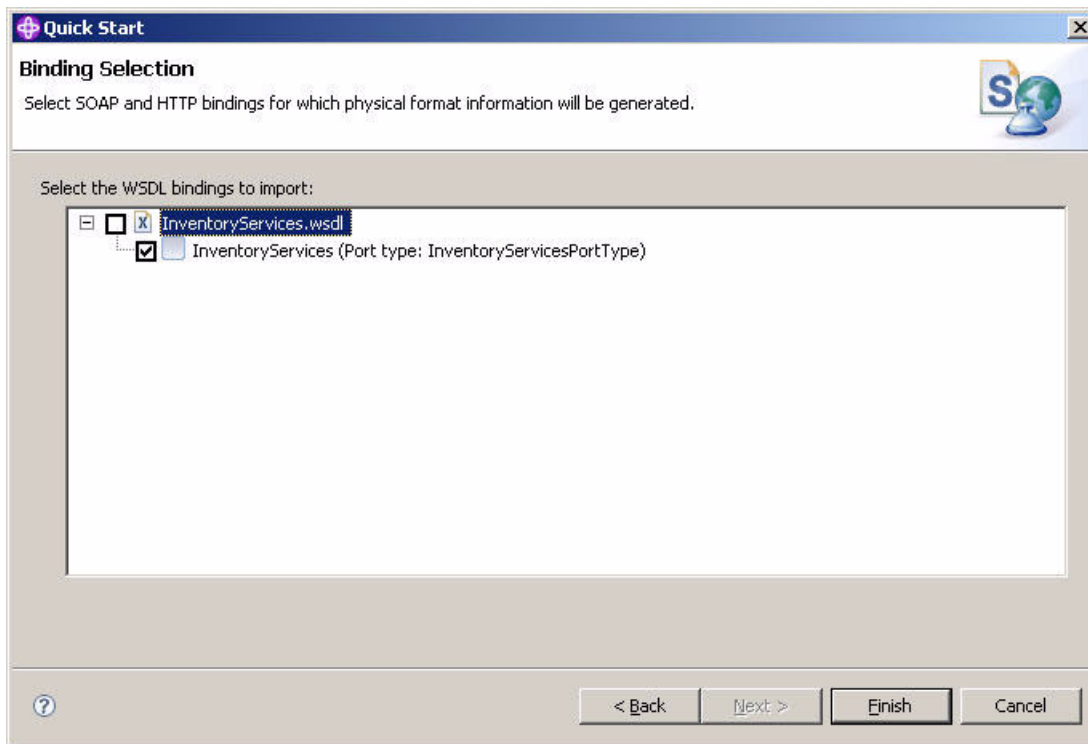


Figure 3-10 Binding Selection

After you click Finish, your toolkit window looks like Figure 3-11.

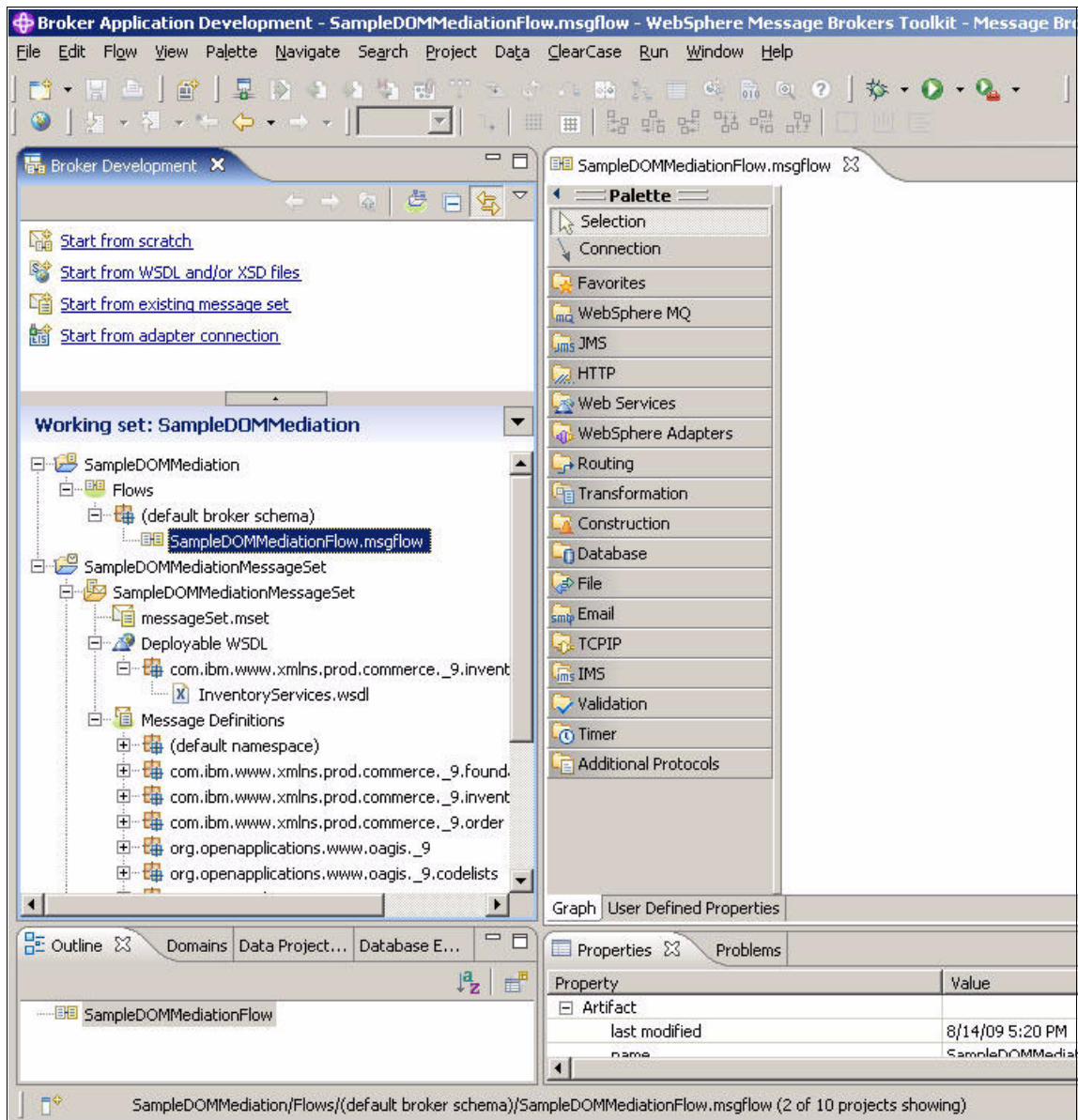


Figure 3-11 Mediation flow

6. Next, you import WSDL for the External OMS simulator. Repeat step 1 on page 102 through step 4 on page 104, and use the values shown in Table 3-2 and Figure 3-12 on page 107, Figure 3-13 on page 108, Figure 3-14 on page 109, and Figure 3-15 on page 110.

Table 3-2 Message set properties

Property	Value
Message flow project name	SampleExtOMS
Message set project name	SampleExtOMSMessageSet
Message set name	SampleExtOMSMessageSet

Quick Start

New Message Broker Application

Set up the basic resources required to develop a Message Broker application using WSDL and XSD files as a starting point.

Message flow project name: SampleExtOMS

Message set project name: SampleExtOMSMessageSet

Message set name: SampleExtOMSMessageSet

Message flow creation

☐ Create a new message flow in my flow project

Message flow name: SampleExtOMSFlow

Working set creation

☐ Create a new working set for these resources

Working set name: SampleExtOMS

< Back Next > Finish Cancel

Figure 3-12 New Message Broker Application

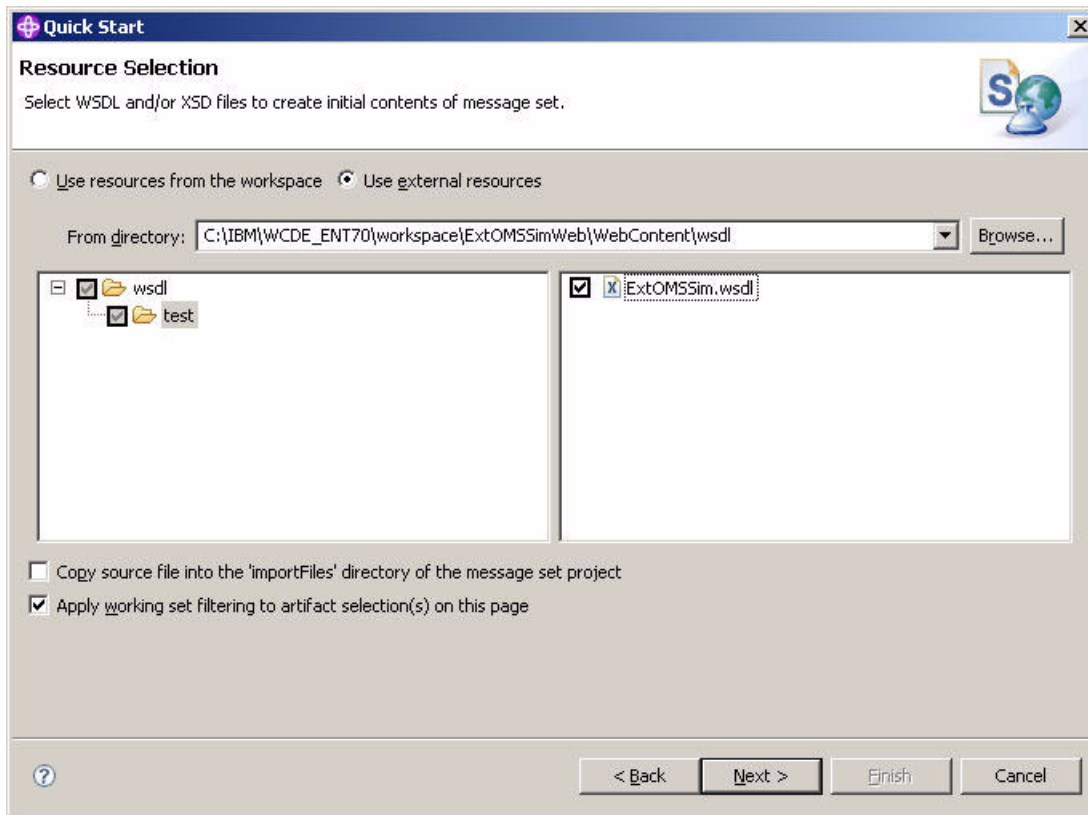


Figure 3-13 Resource Selection

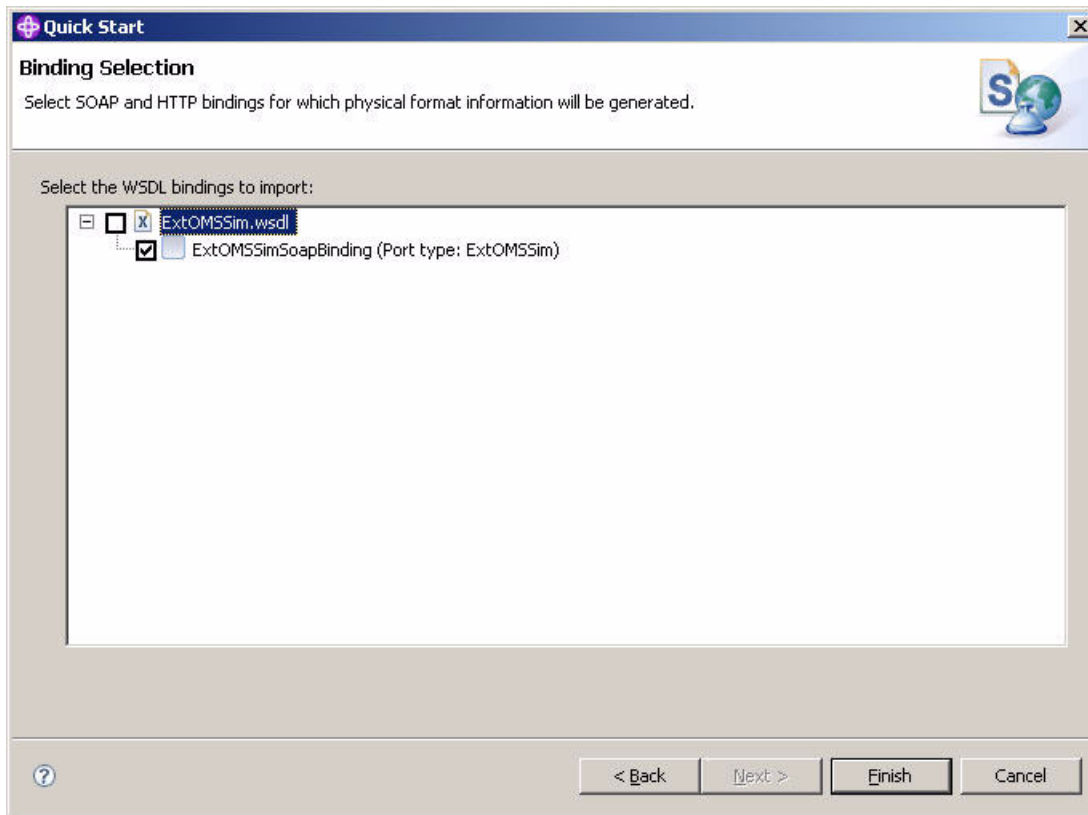


Figure 3-14 Binding Selection

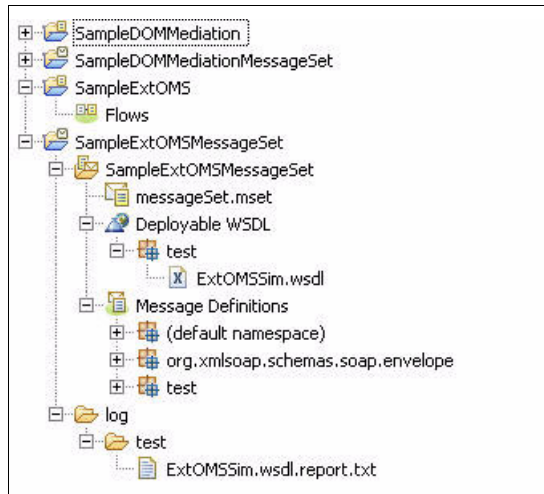


Figure 3-15 SampleDOMMediation

Implementing the mediation flow

After creating the mediation module project, you can implement the message flow for the ReserveInventory service as follows:

1. Open **SampleDOMMediationFlow.msgflow** in the editor.
2. Expand the **SampleDOMMediationMessageSet** node until you get to **InventoryServices.wsd1** (Figure 3-16).

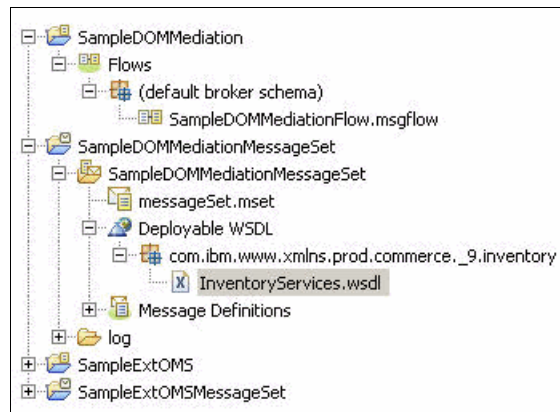


Figure 3-16 InventoryServices.wsd1

3. Drag **InventoryServices.wsdl** into the SampleDOMMediationFlow.msgflow editor. When you drop the WSDL in the editor, it asks for Web service usage configuration. Configure it as shown in Figure 3-17, and click **Next**.

Configure New Web Service Usage

Configure web service usage

Specify the details of how the selected web service will be used in the message flow. Only SOAP HTTP bindings are supported.

Web service usage

☒ Expose message flow as web service

☐ Invoke web service from message flow

Web service parameters

Port type: InventoryServicesPortType

Imported binding: InventoryServices

Service port: InventoryServices

Binding operations:

- ☒ GetInventoryAvailability(GetInventoryAvailability)
- ☒ ChangeInventoryAvailability(ChangeInventoryAvailability)
- ☒ SyncInventoryAvailability(SyncInventoryAvailability)
- ☒ ProcessInventoryRequirement(ProcessInventoryRequirement)

Select All Deselect All

< Back Next > Finish Cancel

Figure 3-17 Configure Web service usage

4. On the next window, select **HTTP nodes**, and click **Finish** (Figure 3-18).

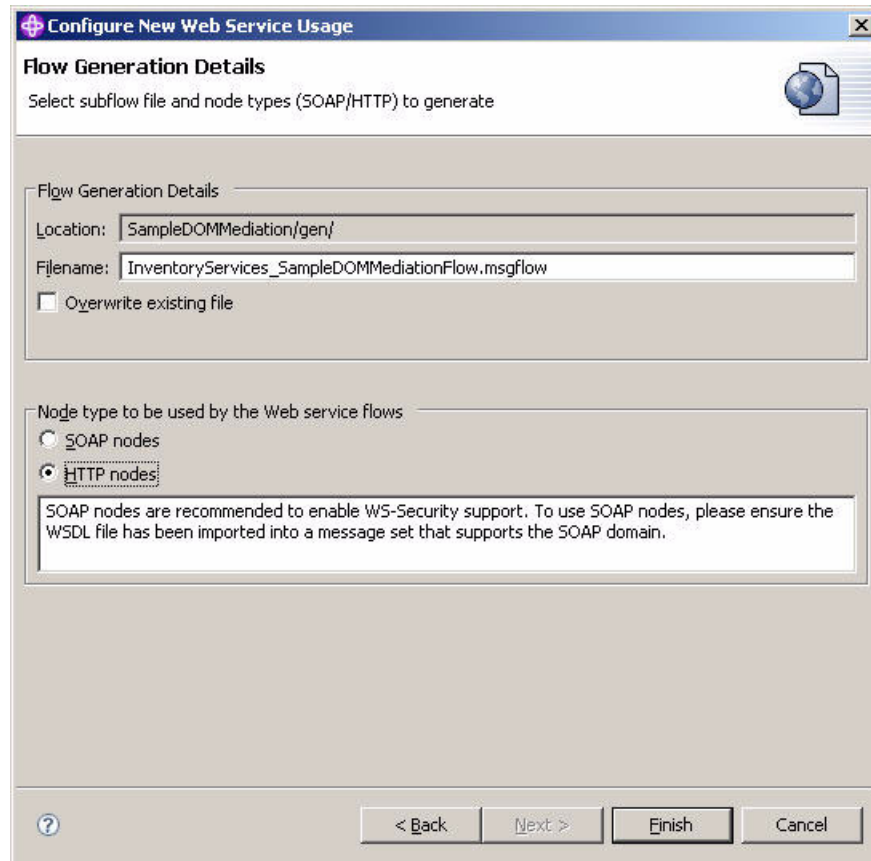


Figure 3-18 Row Generation Details

5. The message flow displays similar to Figure 3-19.

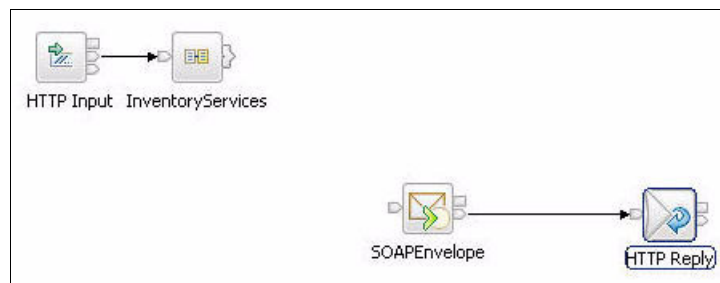


Figure 3-19 Message flow

6. Double-click **InventoryServices** subflow, and it looks similar to Figure 3-20.

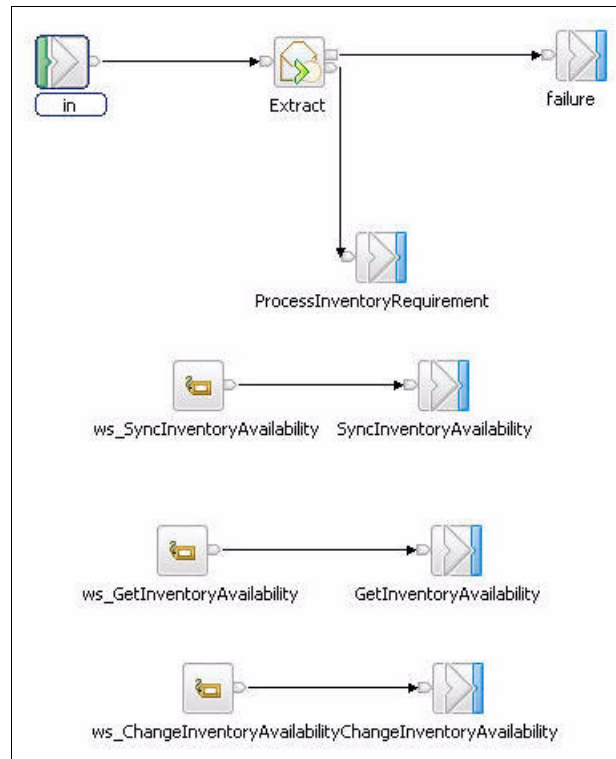


Figure 3-20 Message flow after *InventoryServices* selected

7. In Figure 3-19, select **HTTP Input** node, and modify the properties in properties pane. Set the Path suffix for URL as `http://localhost:7080/InventoryService/processInventory`, where *localhost:7080* is the host name and port where you want to publish this Web service.
8. In Figure 3-20, in *InventoryServices* subflow(*InventoryServices_SampleDOMMediationFlow.msgflow*), rename the *in* node as *InventoryRequest*. Refer to Figure 3-21.

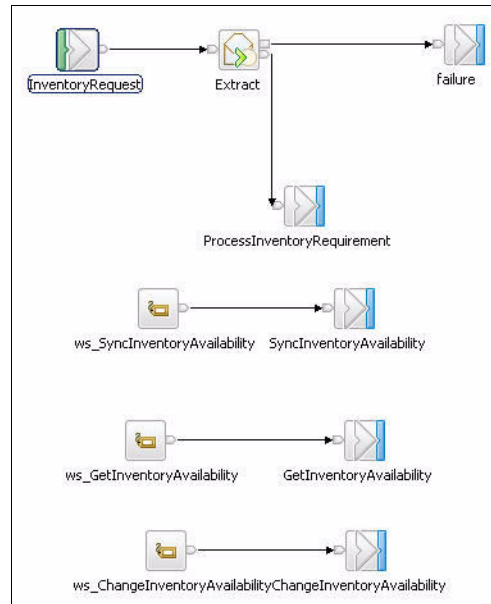


Figure 3-21 Message flow

- Expand **SampleExtOMSMessageSet** Project in the navigator window and navigate to ExtOMSSim.wsdl. Drag this WSDL into SampleDOMMediationFlow.msgflow. Configure the Web service properties as shown in Figure 3-22.

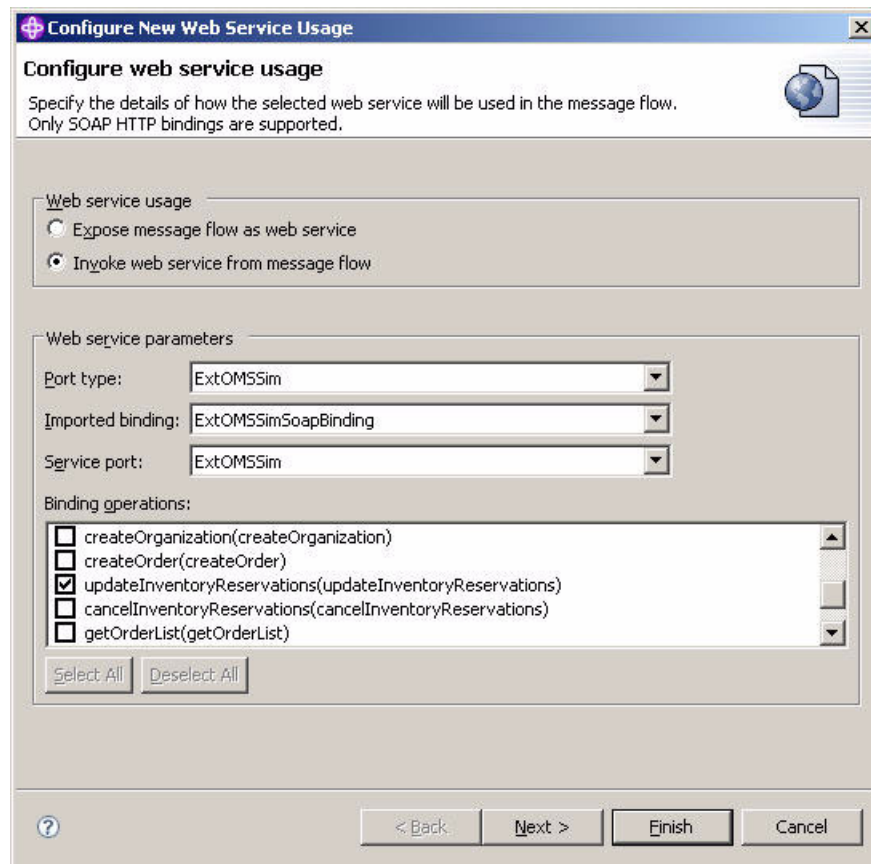


Figure 3-22 Configure Web service usage

10. Click **Next** and select **HTTP nodes** on the next window. Click **Finish**. A subflow `getInventory_ExtOMSSim` is added to `SampleDOMMediationFlow.msgflow` (Figure 3-23).

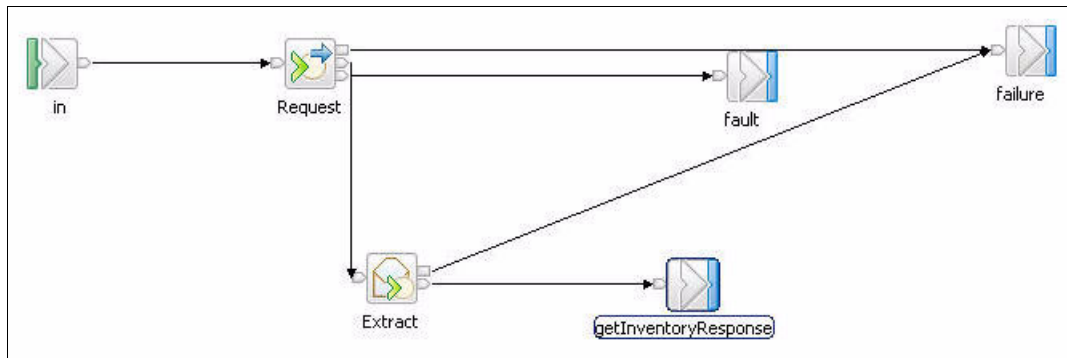


Figure 3-23 Message flow

11. Set the Web service URL property for the request node to the URL of the external OMS simulator service. In our case, it is `http://localhost:9980/ExtOMSSimWeb/services/ExtOMSSim` (Figure 3-24).

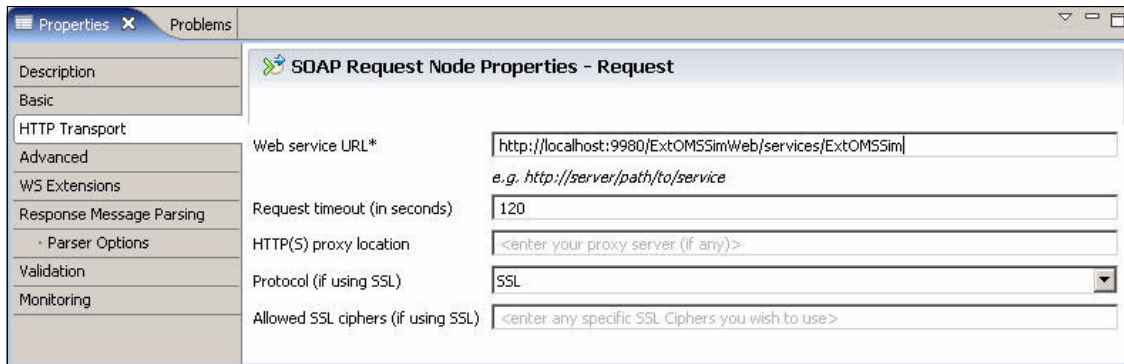


Figure 3-24 SOAP Request Node Properties - Request

Implementing the transformation nodes

There are multiple ways of transforming the messages. It can be done using the mapping node, Java compute node, or XSL transformation. For our example, we have chosen XSL transformation as the way to transform the messages. The mediation flow must transform the input message `ProcessInventoryRequirement` to the message `updateInventoryReservations` and output message `updateInventoryReservationsResponse` to the message `AcknowledgeInventoryRequirement`.

To create the transformation nodes:

1. Create a new XSL. Click **File** \emptyset **New** \emptyset **Other** \emptyset **XSL**. Enter the name of the parent folder as `SampleDOMMediation` and the file name as `ProcessInventoryReqmt_UpdateInventoryReservationsReq.xml` (Figure 3-25). Click **Finish**.

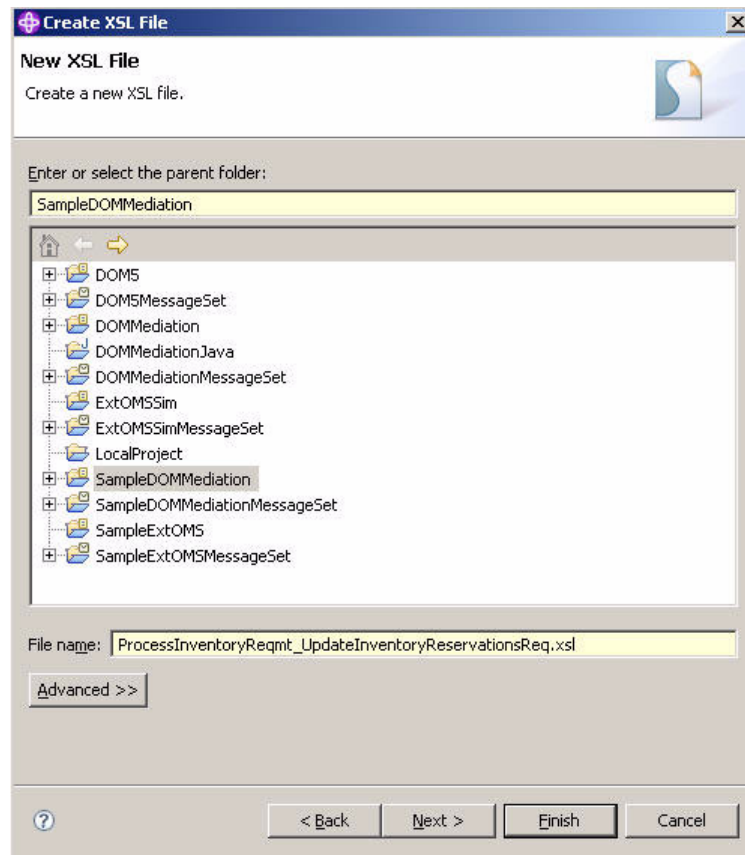


Figure 3-25 New XSL File

2. Add an XSL Transform node from the palette to SampleDOMMediationFlow. Rename the node to ProcessInventoryReqmtToUpdateInventoryReservationsReq. Modify the properties of this node and set the stylesheet name to ProcessInventoryReqmt_UpdateInventoryReservationsReq.xsl. Set Output Message Parsing properties as shown in Figure 3-26.

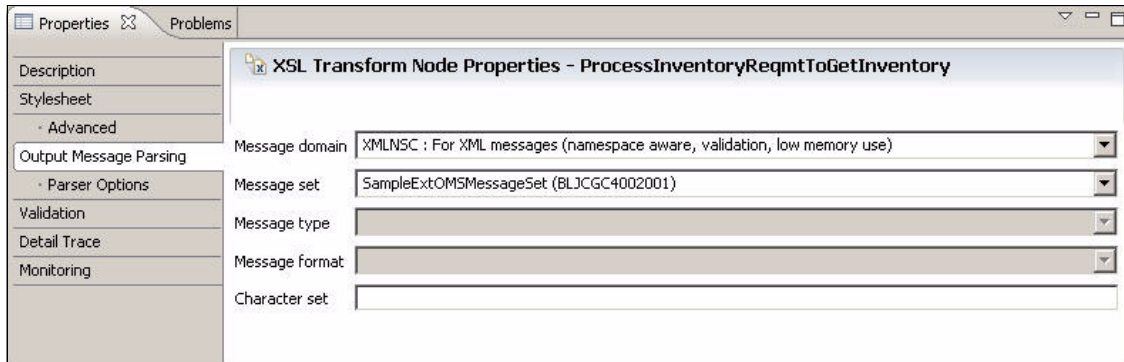


Figure 3-26 XSL Transform Node Properties

3. Implement the XSL ProcessInventoryReqmt_UpdateInventoryReservationsReq.xsl. This XSL transforms the ProcessInventoryRequirement message to updateInventoryReservations message. The sample XSL is given in A.9, “ProcessInventoryReqmt_UpdateInventory ReservationsReq.xsl” on page 477.

4. In SampleDOMMediationFlow connect the OUT terminal of the InventoryServices node to IN terminal of the ProcessInventoryReqmtToUpdateInventoryReservationsReq node. When you click the **OUT** terminal of the InventoryServices node, you will get a Terminal Selection window. Select **ProcessInventoryRequirement**. Refer to Figure 3-27.

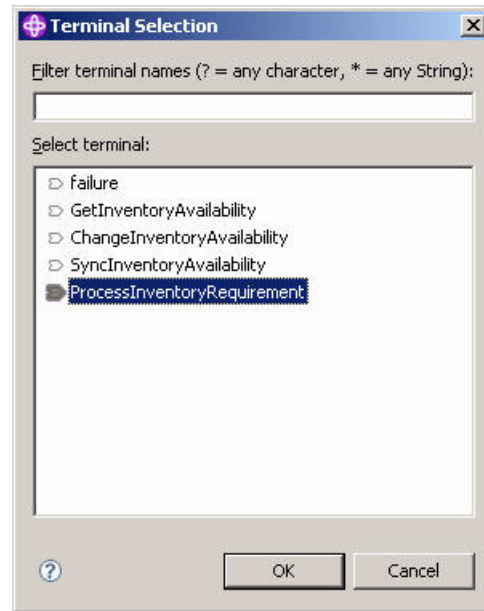


Figure 3-27 Terminal Selection

5. Connect the OUT terminal of ProcessInventoryReqmtToUpdateInventoryReservationsReq to the IN terminal of the getInventory_ExtOMSSim node.

6. Create another XSL, as done in step 1 on page 117. Name the XSL `UpdateInventoryReservationsResp_AcknowledgeInventoryReqmt.xsl`. Add another XSL Transform node to the message flow as in step 2 on page 118. Rename the node `UpdateInventoryReservationsRespToAcknowledgeInventoryReqmt`. Modify the properties of this node and set the stylesheet name to `UpdateInventoryReservationsResp_AcknowledgeInventoryReqmt.xsl`. Set the Output Message Parsing properties as shown in Figure 3-28.

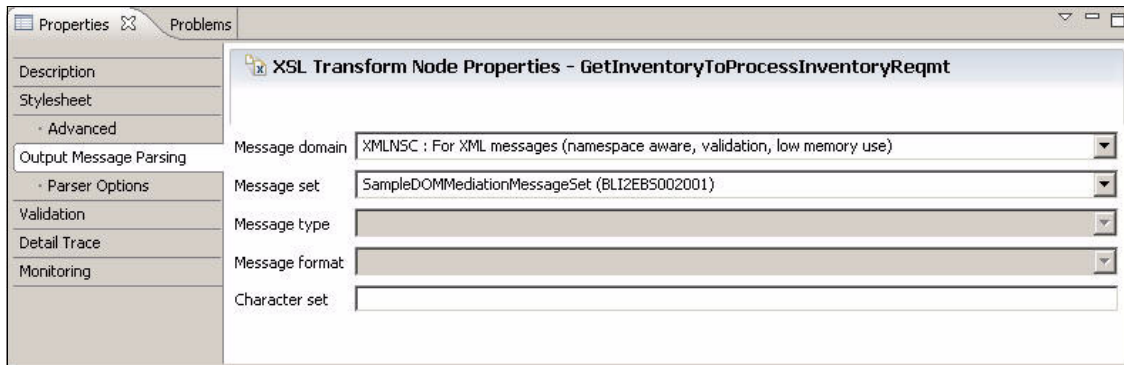


Figure 3-28 XSL Transform Node Properties

7. Implement the XSL `UpdateInventoryReservationsResp_AcknowledgeInventoryReqmt.xsl`. This XSL transforms the `updateInventoryReservationsResponse` message to the `AcknowledgeInventoryRequirement` message. A sample implementation of XSL is given in A.10, “`UpdateInventoryReservationsResp_AcknowledgeInventoryReqmt.xsl`” on page 480.

8. In SampleDOMMediationFlow connect the OUT terminal of the getInventory_ExtOMSSim node to the IN terminal of UpdateInventoryReservationsRespToAcknowledgeInventoryReqmt and the OUT terminal of UpdateInventoryReservationsRespToAcknowledgeInventoryReqmt to the IN terminal of the SOAPEnvelope node. Refer to Figure 3-29.

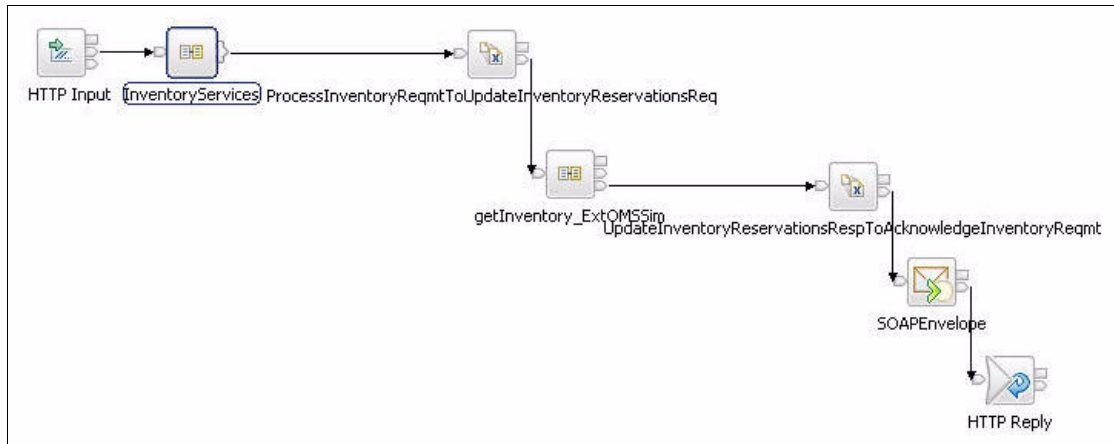


Figure 3-29 Message flow

9. In the properties window for the SOAPEnvelope node, select the **Create New Envelope** check box.

3.3.5 Deploying mediation module

Before starting the deployment process, we must connect the configuration manager (WBRK61_DEFAULT_CONFIGURATION_MANAGER). To connect, right-click **WBRK61_DEFAULT_CONFIGURATION_MANAGER** in the Domains navigator, and select **Connect**.

1. Switch to the Broker Administration perspective.
2. Create a New broker archive, SampleDOMWCS.bar (Figure 3-30):
 - a. Go to **File** \emptyset **New** \emptyset **Message Broker Archive**.
 - b. Select the project as **LocalProject**.
 - c. Enter the name SampleDOMWCS.bar. Click **Finish**.

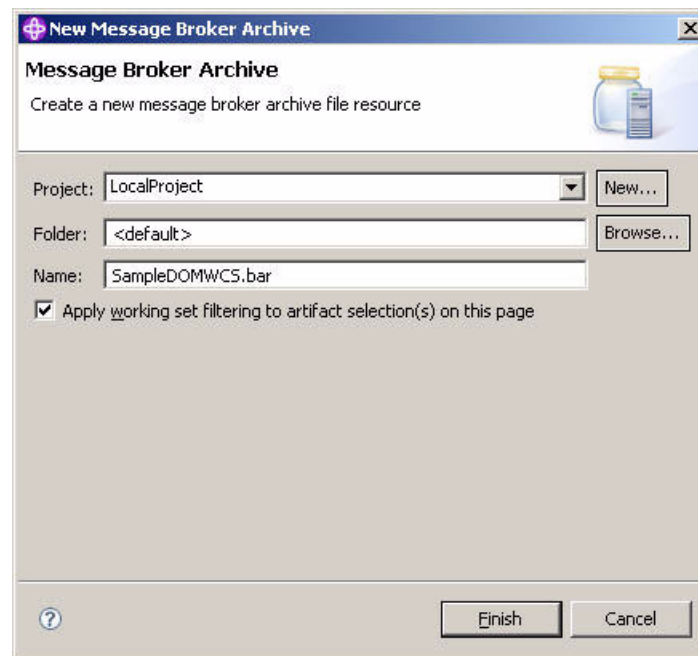


Figure 3-30 Message Broker Archive

3. Open SampleDOMWCS.bar in the editor. Select **SampleDOMMediation** in Filter Working Set. Select the options as shown in Figure 3-32 on page 124, and click **Build broker archive**.

4. Create a Execution Group, Samp1eDOMWCS.
 - a. Go to **File** ∅ **New** ∅ **Execution Group**.
 - b. Select **WBRK61_DEFAULT_BROKER** and for the Execution Group name enter Samp1eDOMWCS. Click **Finish**. Refer to Figure 3-31.

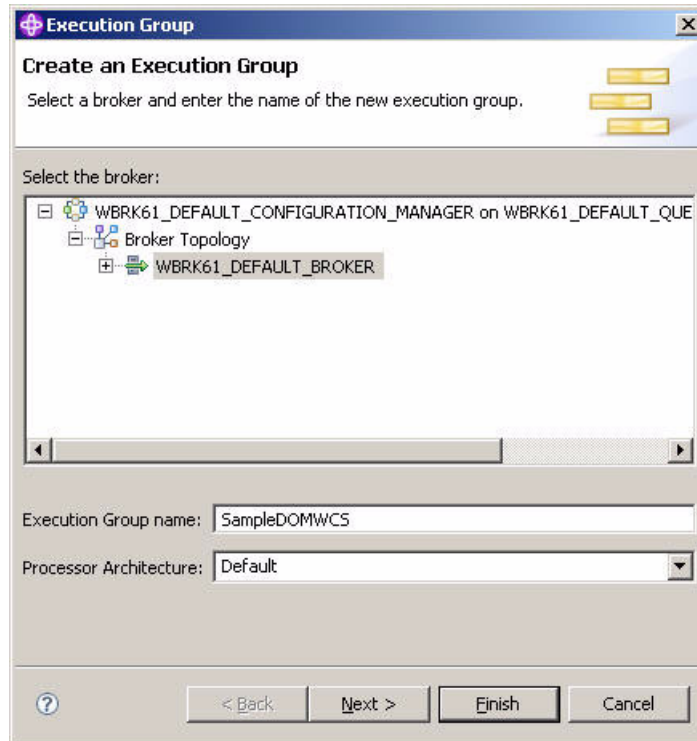


Figure 3-31 Create an Execution Group

5. Right-click **SampleDOMWCS.bar** in the Broker Administration Navigator, and click **Deploy File**. Select **SampleDOMWCS** as the execution group. See Figure 3-32.

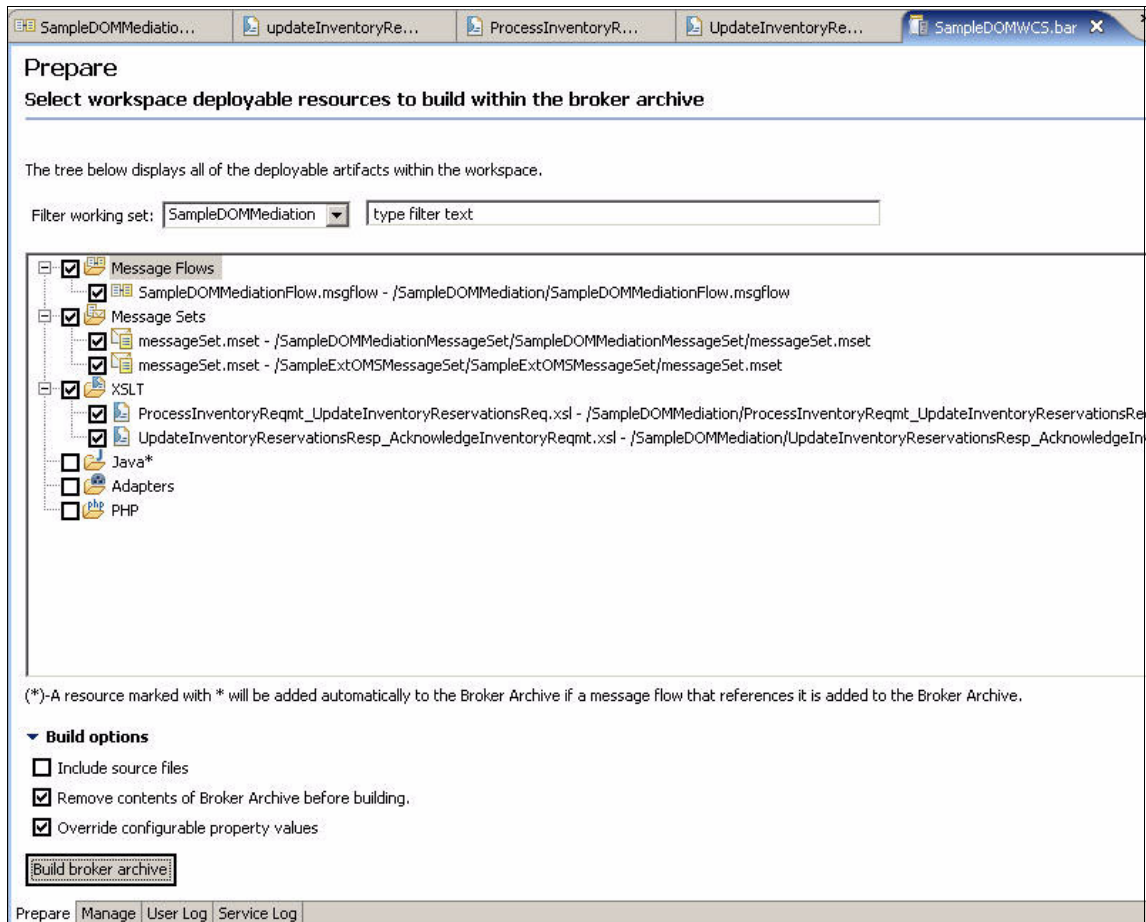


Figure 3-32 Prepare

- After the broker archive deploys, right-click **SampleDOMMediationFlow** in the Domains navigation window, and click **Start** (Figure 3-33).

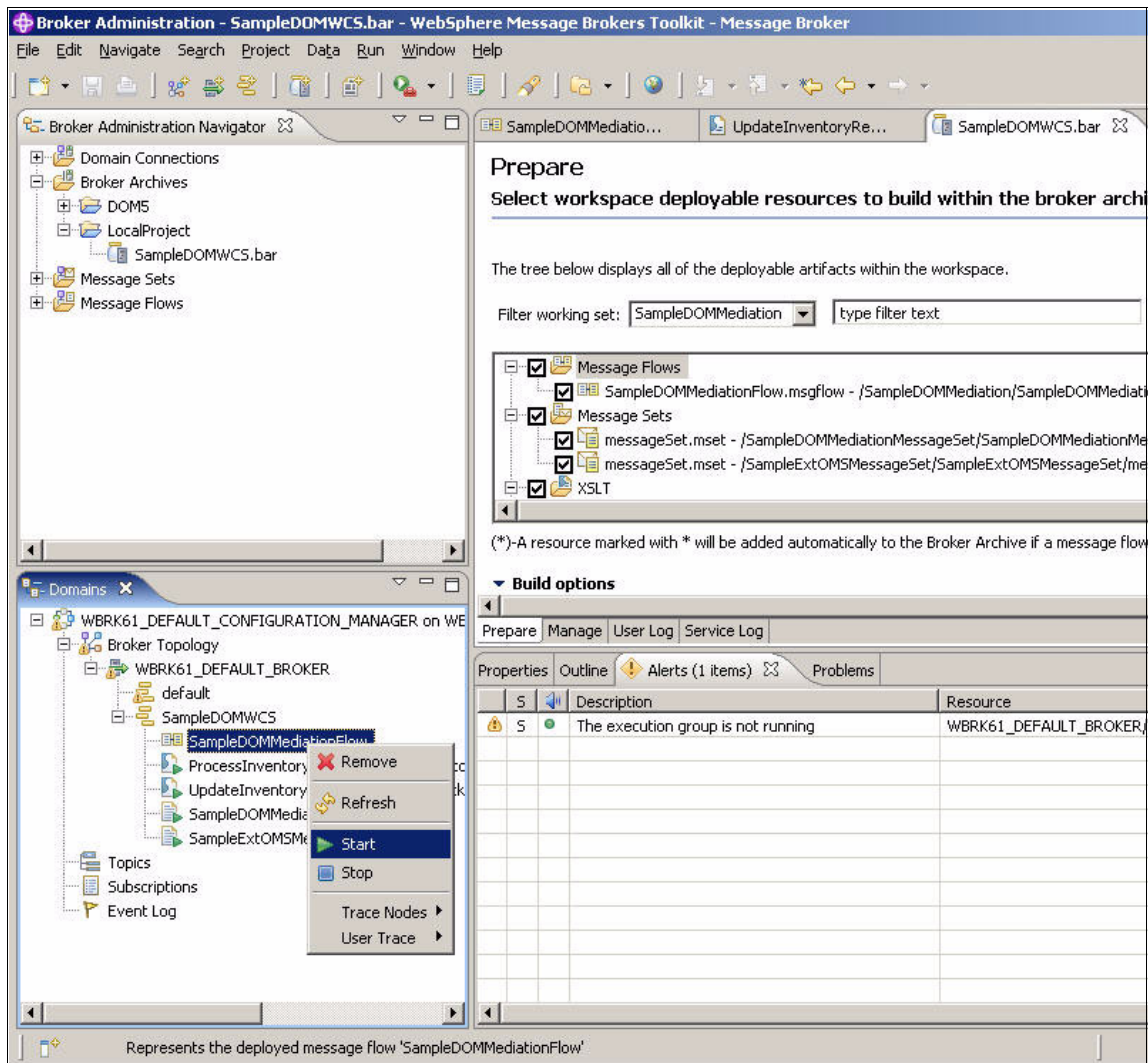


Figure 3-33 Broker Administrator

3.4 Implementing WebSphere Enterprise Service Bus mediation module for DOM integration

You can also perform the implementation steps that are explained in 3.3, “Implementation of WebSphere Message Broker mediation module for DOM integration” on page 98 using WebSphere Enterprise Service Bus and WebSphere Integration Developer. For a detailed step-by-step process and sample code, refer to the *Building a WebSphere Enterprise Service Bus mediation module for DOM Integration* topic in the WebSphere Commerce Version 7 Information Center:

<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.dom-integration.doc/tasks/tsmdombuildmedmod.htm>

3.5 Configuring the DOM integration feature

You must make integration changes in WebSphere Commerce to integrate with the mediation module that was developed in 3.3, “Implementation of WebSphere Message Broker mediation module for DOM integration” on page 98, which in turn integrates with External DOM simulator.

To configure the DOM integration feature:

1. Start the WebSphere Commerce server.
2. Open the Administration console.
3. On the Site/Store Selection page, select **Site**.
4. Select **Configuration** \varnothing **Transports**.
5. Click **Add**.
6. Select **Web Services (HTTP)**.
7. Click **Add**.
8. Select **Configuration** \varnothing **Message Types**.
9. Create the message type configuration listed in Table 3-3.

Table 3-3 Message types

Property	Value
Message type	com.ibm.commerce.inventory.external
Transport	WebServices (HTTP)

Property	Value
Device Format	webservices
URL	http://localhost:7080/InventoryService/processInventory

10. Close the Administration Console.
11. Connect to the database.
12. Execute the following SQL statement to use an external system for inventory and order processing:

```
UPDATE STORE SET INVENTORYSYSTEM=-5 WHERE STORE_ID IN (SELECT
STOREENT_ID FROM STOREENT WHERE IDENTIFIER='Store_Identifier' );
```

Where Store_Identifier is the value of your store identifier. For example, 'Madisons' for your consumer direct store.
13. Replace the contents of the wc-component-client.xml file in workspace_dir/wc/xml/config/com.ibm.commerce.inventory.external/wc-component-client.xml with the contents in Example 3-5.

Example 3-5 The wc-component-client.xml file

```
<?xml version="1.0" encoding="UTF-8"?>
<_config:DevelopmentClientConfiguration
xmlns:_config="http://www.ibm.com/xmlns/prod/commerce/foundation/config"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.ibm.com/xmlns/prod/commerce/foundation/config
../xsd/wc-component-client.xsd">
  <_config:invocationService>
    <_config:invocationBinding
      bindingImpl="com.ibm.commerce.foundation.internal.client.services.
      invocation.impl.JCAInvocationBindingImpl"></_config:invocationBinding>
    <_config:action name="GetInventoryAvailability"
      asynchronous="false"></_config:action>
    <_config:action name="ProcessInventoryRequirement"
      asynchronous="false"></_config:action>
  </_config:invocationService>
</_config:DevelopmentClientConfiguration>
```

14. Restart the WebSphere Commerce server.

After completing the steps in 3.3, “Implementation of WebSphere Message Broker mediation module for DOM integration” on page 98, and 3.5, “Configuring the DOM integration feature” on page 126, browse to the Madisons Starter Store product pages, and add items to the cart. When you try to check out from the

cart, a call is made to the ProcessInventoryRequirement outbound service with the action code ReserveInventory.

Because we implemented the ReserveInventory flow in WebSphere Message Broker, the External OMS simulator is invoked and a response with allocated inventory is returned. Refer to the following sections:

- ▶ A.1, “ProcessInventoryRequirement with action code ReserveInventory request” on page 456, for a sample request
- ▶ A.2, “ProcessInventoryRequirement with action code ReserveInventory response” on page 460, for a sample response



Web 2.0 storefront

This chapter describes the Madisons Starter Store (Web 2.0 storefront) and explains how to integrate MapQuest with the Store Locator feature in IBM WebSphere Commerce V7.

This chapter includes the following topics:

- ▶ WebSphere Commerce Web 2.0 store overview
- ▶ Buy online, pick up in store
- ▶ The Store Locator feature
- ▶ MapQuest integration for Store Locator
- ▶ MapQuest Geocoding

4.1 WebSphere Commerce Web 2.0 store overview

IBM WebSphere Commerce V7 supplies two Web 2.0 sample starter stores:

- ▶ The Madisons Starter Store, a business-to-consumer model
- ▶ The Elite starter store, a business-to-business model

Both of these sample stores include features that shoppers might want, for example placing orders by drag, fast finder with slide bars for product attributes, and so forth.

These stores support both distributed order management (referred to as *DOM* throughout this book) integration solution and buy online, pick up in store (referred to as *BOPIS* in this book).

4.1.1 Web 2.0 in WebSphere Commerce

Web 2.0 introduces Web features other than Web 1.0. WebSphere Commerce exploits these features in Web 2.0 starter stores to attract more customers to the storefront so that more orders are placed in the Web 2.0 store.

WebSphere Commerce uses the Dojo Ajax and events API to provide a framework that meets all the Ajax requirements for storefront development. For more information about Dojo framework, refer to:

<http://www.dojotoolkit.org/docs>

IBM WebSphere Commerce V7 extends the Dojo framework by providing the following APIs and widgets:

- ▶ APIs

- `wc.service.declare (initProperties)`

This function declares a new Ajax service with the specified ID. A service is a server URL that performs a server object create, update, delete or other server processing in WebSphere Commerce. By using this API, you will have the capability to call a WebSphere Commerce server URL using Ajax whenever it is needed in your JavaScript code.

- `wc.service.invoke (serviceId, parameters)`

This function finds the registered service with the specified service ID and invokes the service through Ajax using the specified parameters.

- `wc.service.getServiceById (serviceId)`

This function gets the service that was declared under the specified identifier. If the service was not declared then this function will return "undefined".

- `wc.render.declareRefreshController (initProperties)`

This function declares a new refresh controller and initializes it with the specified initialization properties. The initialization properties are “mixed in” with the new refresh controller’s properties.

A refresh controller controls refresh area widgets. It listens to changes in the render context and changes to the model and decides if the registered refresh areas should be updated.

- `wc.render.getRefreshControllerById (ID)`

This function returns the refresh controller that was declared under the specified identifier. If the refresh controller was not declared then this function will return "undefined".

- `wc.render.declareContext (ID, properties, updateContextURL)`

This function declares a new render context and initializes it with the specified render context properties. The update context URL is used to report changes to the render context to the server.

- `wc.render.getContextById (ID)`

Get the render context that was declared under the specified identifier. If the render context is not declared, then this function returns *undefined*.

- `wc.render.updateContext (ID, updates)`

This function retrieves the render context with the specified ID and applies the updates found in the specified *updates* object.

► Widgets

- `wc.widget.RefreshArea`

The refresh area widget is used to wrap a DOM node that might need to be refreshed by replacing the `innerHTML` property with fresh HTML loaded from the server. A refresh area widget is associated with a registered refresh controller that handles listening for events that will require this widget to be refreshed.

- `wc.widget.RangeSlider`

This widget is a widget that defines a two handle slider which can be used to drive a price range selection in the storefront. It is being used in the fast finder page.

- `wc.widget.ScrollablePane`

A scrolling thumbnail picker widget that can accept any content and it can scroll these content. The `ScrollablePane` is the widget which provides scrolling effect for the items or images which is part of the content inside the widget. Each item in the scrollable should be in side a `ContentPane` widget. User can scroll the items manually as well as auto. By passing the `autoScroll` paramenter as true items will scroll automatically. There are two basic control buttons to allow users to manually scroll the contents left and right as desired.

The Madisons starter store uses these Web 2.0 APIs. You can use the Madisons Starter Store as the base store for a customized Web 2.0 store. For more information, refer to 4.1.2, “Madisons Starter Store” on page 132.

For more information about WebSphere Commerce Web 2.0 technology, refer to *WebSphere Commerce Best Practices in Web 2.0 Store*, SG24-7647.

4.1.2 Madisons Starter Store

The Web 2.0 Madisons Starter Store is enhanced in IBM WebSphere Commerce V7. For more detailed information about these enhancements, refer to 2.1, “Enhanced Madisons Starter Store” on page 30.

As mentioned in “New Change Flow options” on page 40, the Web 2.0 features in Madisons Starter Store are configurable. The store administrators can enable or disable the Web 2.0 features in the WebSphere Commerce Accelerator tool.

To enable the Web 2.0 feature Product drag, you need to first enable the Compare zone and Mini shopping cart feature.

Figure 4-1 illustrates how to enable the Compare zone feature. You access these enablement options by selecting **Store** ∅ **Change Flow** from the menu, and clicking **Catalog** in the notebook.

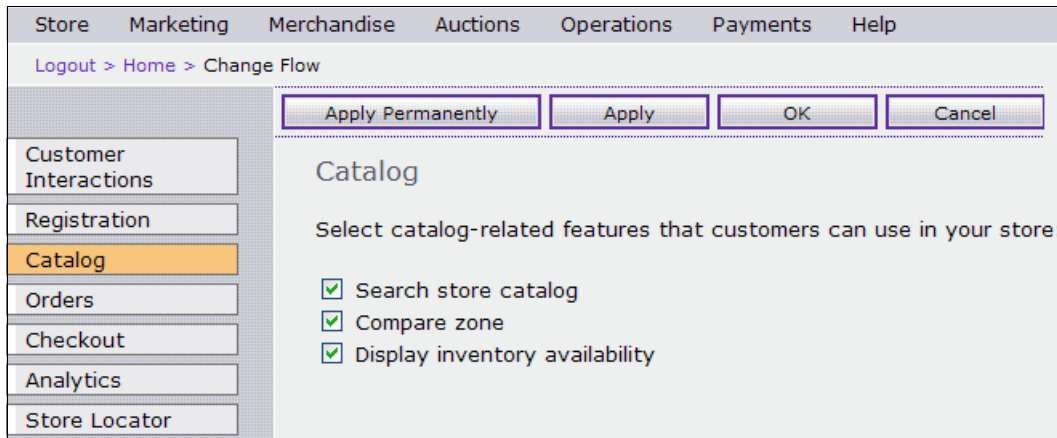


Figure 4-1 Enable compare zone in WebSphere Commerce Accelerator

Figure 4-2 shows how to enable the mini shopping cart page in WebSphere Commerce Accelerator. You can access these enablement options by selecting **Store** ∅ **Change Flow** from the menu, and clicking **Orders** in the notebook.

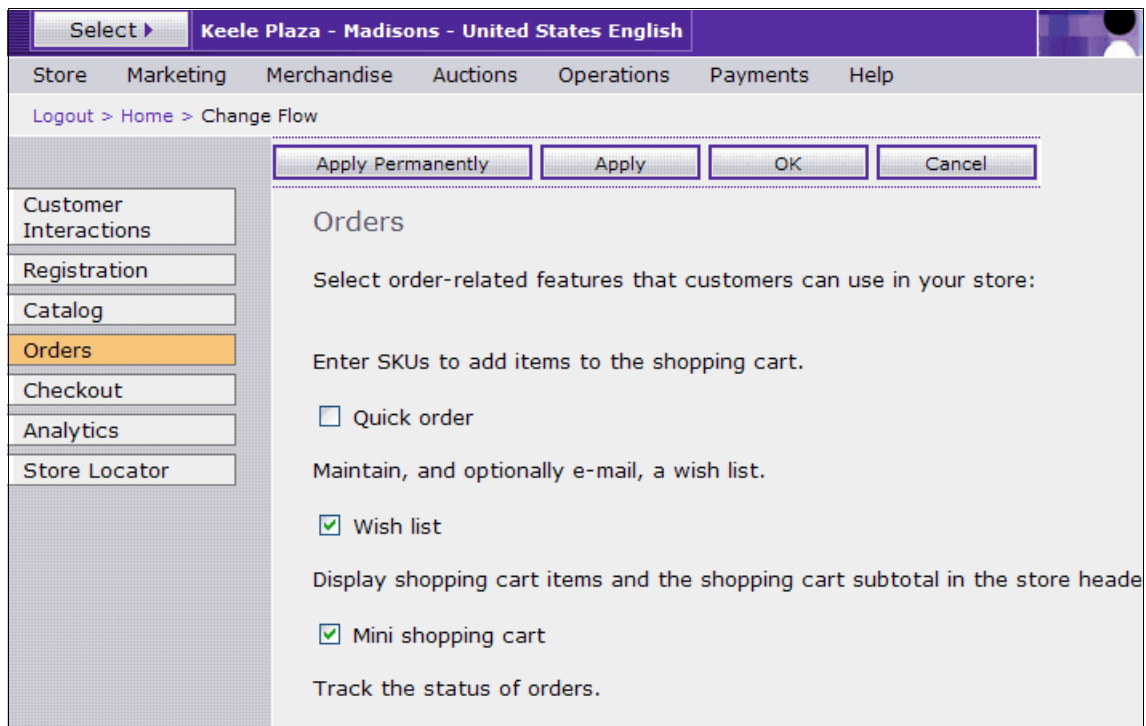


Figure 4-2 Enable mini shopping cart in WebSphere Commerce Accelerator

4.2 Buy online, pick up in store

In this book, we use the term *BOPIS* to represent *buy online, pick up in store*. The BOPIS feature ships with the IBM WebSphere Commerce V6 Feature Enhancement Pack 5, which supports the DOM inventory. With IBM WebSphere Commerce V7, the BOPIS feature is enhanced to support all the inventory systems in WebSphere Commerce, including available to promise (ATP) inventory, non-ATP inventory, and DOM.

4.2.1 BOPIS overview

In IBM WebSphere Commerce V6 Feature Enhancement Pack 5, the DOM feature must be enabled to use the BOPIS. For information about how to enable the DOM feature in Feature Enhancement Pack 5, refer to:

http://publib.boulder.ibm.com/infocenter/wchelp/v6r0m0/index.jsp?topic=/com.ibm.commerce.dom-integration.doc/tasks/tsmbopisinstall_dup.htm

In IBM WebSphere Commerce V7, BOPIS is a base feature of the Madisons Starter Store and the Elite starter store without extra installation and configuration steps.

BOPIS enables shoppers to browse the online category. When the shopper check out the items, instead of asking the online store to ship the items, the shopper can select a physical store and pick up the items from there.

BOPIS also provides more choices for payment method when checking out. The shoppers can pay online or can choose to pay in store when they pickup the merchandise.

By selecting pickup in store, shoppers can save shipping costs that are associated with a delivery and can choose a time that is convenient to pick up the merchandise rather than having to wait for a delivery. Shoppers can choose a physical store that is convenient for pickup.

By enabling BOPIS, the retailer provides this convenience option to shoppers, which also has the benefit of shoppers physically visiting a brick-and-mortar store, where additional purchases can take place.

4.2.2 BOPIS in Madisons Starter Store

In a typical end-to-end BOPIS scenario, an online shopper takes the following general steps to complete the shopping process:

1. The online shopper browses the product catalog online.
2. The online shopper adds a product into the Shopping Cart and checks out.
3. When checking out, the online shopper chooses to pick up the product in the store instead of shipping the product.
4. When the online shopper selects the pick up in store option, there are options for payment, including pay in the store.
5. The online shopper submits the order.
6. The online shopper goes to the store and pays for the merchandise and picks up the merchandise in store.

Here is a more detailed explanation of each step regarding this end-to-end scenario:

1. The online shopper browses the categories.

The online shopper goes to the Madisons Starter Store storefront and does some browsing. The shopper is interested in the lounge chairs in the furniture category and clicks the lounge chairs category for more detailed information.

Then, the products in the lounge chairs category are listed as shown in Figure 4-3.

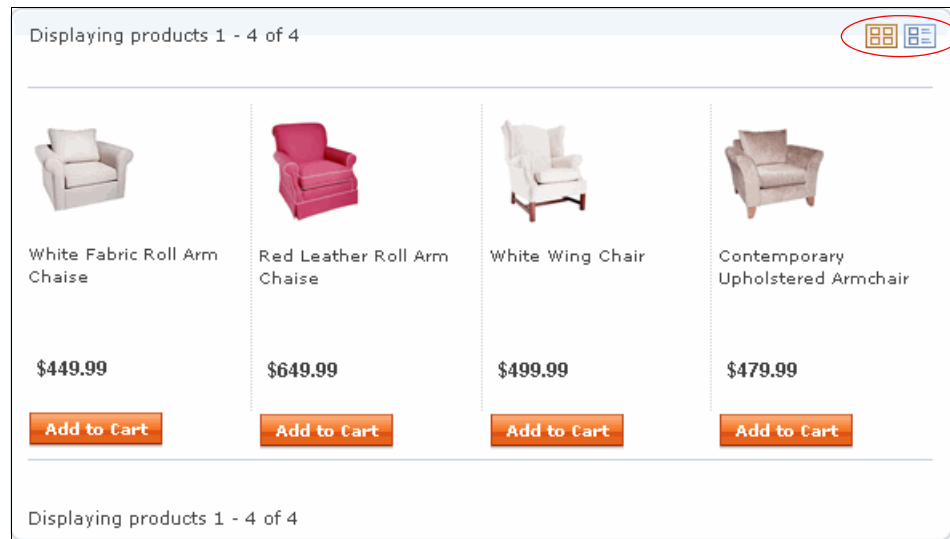


Figure 4-3 Products of lounge chairs category listed

Note: The online shopper can click the icons in the top, right of the page to change how these products are listed. They can be listed by details or in a table.

In this page, the shopper can click **Add to Cart** to add one item into the Shopping Cart. However, this page does not display detailed information about the product.

When browsing the products, the online shopper can complete the following tasks:

a. The online shopper selects to view quick information.

When the online shopper moves the mouse pointer over the item, a window opens that includes the Quick Info button. The shopper clicks this button to view the quick information about this product.

The quick information window offers more options to the shopper, as shown in Figure 4-4. From this window, the online shopper can choose Add to Cart, Add to Wish List, or Add to Compare.



Figure 4-4 Quick information about an item

b. The online shopper views the product detail page.

To get more information, the online shopper clicks the **more info** link to view the product details page. The shopper can also choose to close this window first and then click the product icon. In either case, the shopper is directed to the product detail page in Figure 4-5.



Figure 4-5 Product detail page

In this example, the shopper can view more detailed information about this white fabric roll arm chair. There are detailed descriptions of this chair in the Description tab. In addition, any attachments for this chair, such as more pictures, display in the Attachments tab.


In the Check Store Availability area, by default, the inventory availability of the online store displays. In this scenario, the chair is in stock.

c. The online shopper adds more stores.

The shopper now can click Show Availability for the in-store availability. If the shopper did not select any favorite store before, the page is redirected to the Store Locator page, where the shopper can select favorite stores.

In the top half of the Store Locator page, the favorite stores previously selected by the shopper display, as shown in Figure 4-6. The shopper can click the Remove button to remove some of them from the favorite store list.

Check availability at a store near you



White Fabric Roll Arm Chaise

Price: \$449.99

Your Store List



STORE NAME AND ADDRESS	HOURS	
Calgary Circle Mall 300 MacLeod Tr Calgary, Alberta T2G 5R1 511.513.5789	Mon-Fri: 10am - 9pm Sat: 9am - 7pm Sun: 11am - 6pm	 Remove
Calgary Mall 1025 Cameron Ave SW Calgary, Alberta T2T 0K4 367.666.6666	Mon-Fri: 10am - 9pm Sat: 9am - 7pm Sun: 11am - 6pm	 Remove

Figure 4-6 The store list in Store Locator top half page

In the bottom half of the Store Locator page, the shopper can add more stores to their favorite store list, as shown in Figure 4-7.

Store Locator

Select a location to find a store near you.

Country:

State/Province:

City:

Canada

Manitoba

Winnipeg

GO

Store Locator Results

Select one or more stores to check product availability.

STORE NAME AND ADDRESS	HOURS	SELECT STORE
Breakfast Place 456 Backwater Bay Winnipeg, Manitoba R2N 1M7 564.213.2589	Mon-Fri: 10am - 9pm Sat: 9am - 7pm Sun: 11am - 6pm	<div>Add to store list</div>
Winnipeg Mall 501 Jessie Ave Winnipeg, Manitoba R3L 0P7 367.777.7777	Mon-Fri: 10am - 9pm Sat: 9am - 7pm Sun: 11am - 6pm	<div>Add to store list</div>

Add a store location to your store list.

Continue Shopping

Figure 4-7 Select the favorite stores in Store Locator page

The shopper can select the country, state, and city and click **Go** to find a store list in this area. Clicking the **Add to store list** button adds the store to the store list.

In this scenario, the shopper adds Breakfast Place and Winnipeg Mall into the favorite store list.

d. The online shopper selects to show the inventory availability.

By clicking **Continue Shopping**, the shopper goes back to the product detail page. Then, when the shopper clicks **Show availability**, the inventory availability of the stores displays in the store list, as shown in Figure 4-8. In this sample, there are sufficient inventories in Calgary Mall and Calgary Circle Mall, but the item is out-of-stock in Breakfast Place and Winnipeg Mall.

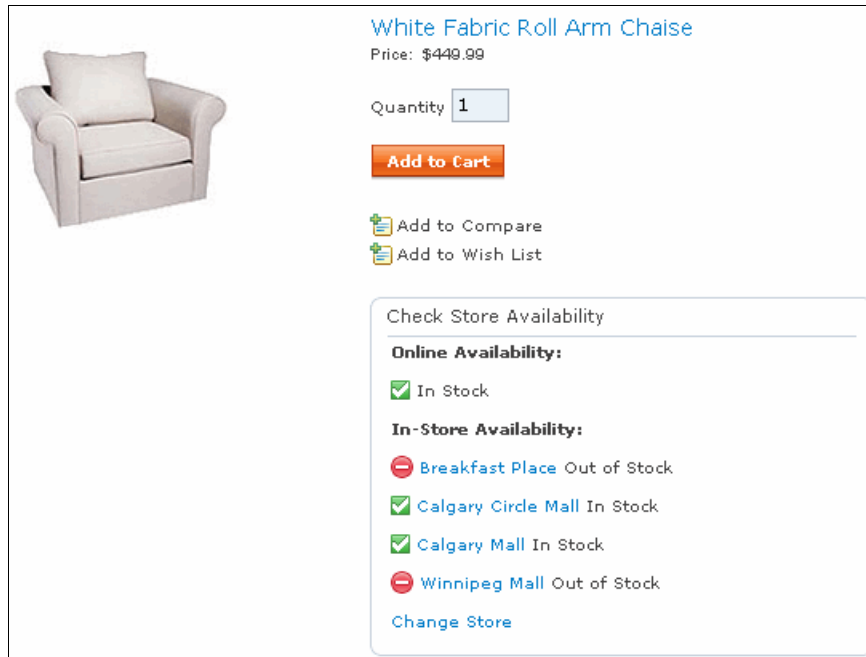


Figure 4-8 Product detail page: Show store inventory availability

2. The online shopper adds the product to the Shopping Cart.

The online shopper clicks the **Add to Cart** button, and the product is added successfully into the shopper's shopping cart. Then, the following message displays in the product detail page to indicate to the shopper that the operation is successful:

The item has been successfully added to your shopping cart.

Note: Because we enabled the *Ajax add to shopping cart* feature for the Madisons Starter Store, the online shopper added successfully the product into the Shopping Cart without leaving the product detail page.

Now the Shopping Cart icon in the top, right shows one item. When he shopper moves the mouse pointer over the item, the mini shopping cart pop-up window displays, as shown in Figure 4-9.

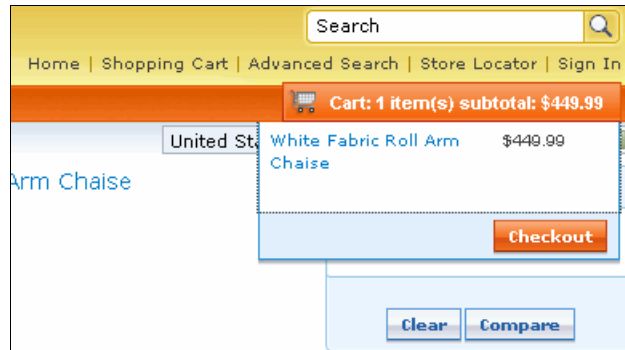


Figure 4-9 The mini shopping cart window

3. The online shopper checks out and chooses to pick up the product in the store instead of shipping the product.

The online shopper clicks the **Checkout** button and goes to the Shopping Cart page to check out, as shown in Figure 4-10. In this page, the shopper can select to continue shopping online or to pick up at store. The shopper can also use this page to update the item quantity or to remove the items. There is also an option for the shopper to input promotion codes.

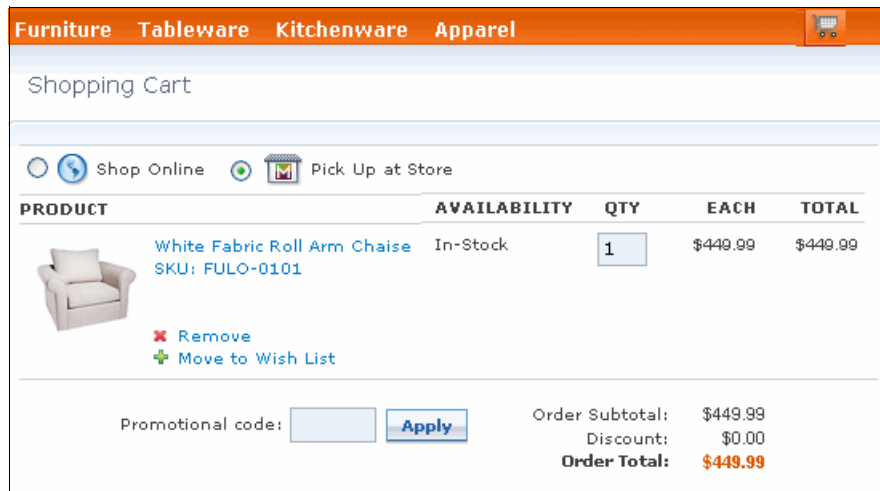


Figure 4-10 Select Pick Up at Store

In this scenario, the shopper selects the “Pick Up at Store” option.

If the shopper did not log on to the store yet, log on options display in the same page, as shown in Figure 4-11.

<p>New Customer & Guests</p> <p>Checkout without signing in</p> <p>You can make your purchases from Madisons without signing in.</p> <p>You will be given the option to register during the checkout steps.</p> <p>Continue Checkout</p>	<p>Returning Customers</p> <p>Sign in for quick checkout</p> <p>Username:</p> <input type="text"/> <p>Password:</p> <input type="password"/> <p>Forgot your password?</p> <p>Sign in & Checkout</p>
---	--

Figure 4-11 Log on options

If the shopper is a registered customer in this store, the shopper can enter a user name and password and then click **Sign in & Checkout**.

4. The online shopper selects a store to pick up and a payment option.
- In this scenario, the shopper clicks **Continue Checkout** as a guest shopper. Then, the online shopper selects a store and payment option as follows:
- The store selection page displays so that the online shopper can choose a store in which to pick up the merchandise. For the four favorite stores listed, the shopper selects **Calgary Mall**, as showed in Figure 4-12, and clicks **Continue** to continue with checkout process.










Your Store List			Hide 
You have selected to pick up your order at the following store location:			
STORE NAME AND ADDRESS	HOURS	AVAILABILITY	
<input type="radio"/> Breakfast Place 456 Backwater Bay Winnipeg, Manitoba R2N 1M7 564.213.2589	Mon-Fri: 10am - 9pm Sat: 9am - 7pm Sun: 11am - 6pm	 Out of Stock	 Remove
<input type="radio"/> Calgary Circle Mall 300 MacLeod Tr Calgary, Alberta T2G 5R1 511.513.5789	Mon-Fri: 10am - 9pm Sat: 9am - 7pm Sun: 11am - 6pm	 In Stock	 Remove
<input checked="" type="radio"/> Calgary Mall 1025 Cameron Ave SW Calgary, Alberta T2T 0K4 367.666.6666	Mon-Fri: 10am - 9pm Sat: 9am - 7pm Sun: 11am - 6pm	 In Stock	 Remove
<input type="radio"/> Winnipeg Mall 501 Jessie Ave Winnipeg, Manitoba R3L 0P7 367.777.7777	Mon-Fri: 10am - 9pm Sat: 9am - 7pm Sun: 11am - 6pm	 Out of Stock	 Remove

Figure 4-12 Select a store

- b. The online shopper selects **Pay in-store**.

On the address page, the shopper can choose whether to pay for the merchandise in store. If the pay in store option is not selected, the shopper must input a billing address. The shopper selects **Pay in-store** as shown in Figure 4-13, and clicks **Next**.

Furniture Tableware Kitchenware Apparel	
Shopping Cart Store Selection Address Shipping & Billing	
1. Billing Address <input checked="" type="checkbox"/> Pay in-store Your payment is made at the store location.	2. Store Address Calgary Mall 1025 Cameron Ave SW Calgary Alberta Canada T2T 0K4 367.666.6666
Back Next Proceed to your Shipping & Billing Method	

Figure 4-13 Store address and pay in-store selection

- c. The online shopper goes to the shipping and billing page, shown in Figure 4-14.

Now the online shopper goes to the shipping and billing page, which displays the address of the Calgary mall. At this point, the shopper can still change the store when needed by clicking the **Change Store** button, which directs the shopper back to the store selection page.

This page also displays that the pay in store option is selected for the payment method. To change the payment option, the online shopper needs to click **Change** button in the Payment Information section to go back to the address page.


If all the information is correct, the online shopper clicks **Next** to proceed.

Shipping Information

Your items will be picked up at the following location:

Calgary Mall
1025 Cameron Ave SW
Calgary Alberta
Canada T2T 0K4
367.666.6666

Change store

Product	Availability	Qty	Each	Total
 <div>White Fabric Roll Arm Chaise SKU: FULO-0101</div> <div><div>Remove</div><div>Move to Wish List</div></div>	In-Stock	1	\$449.99	\$449.99

Save 20% on Furniture! (\$90.00)

Promotional code:

Apply

Order Subtotal: \$449.99
Product Discounts: (\$90.00)
Discount: (\$36.00)
Tax: \$0.00
Shipping: \$0.00
Shipping Tax: \$0.00
Order Total: \$323.99

Billing Information

Currently selected payment method:

Your payment is made at the store location.

Change

Back

Next

 Proceed to your Order Summary.

Figure 4-14 Shipping and billing page

- d. A summary of the current order displays in this page, as shown in Figure 4-15.


Shopping Cart Store Selection Address Shipping & Billing Method Order Summary					
Shipping Information					
Shipping Address: Calgary Mall Calgary Mall 1025 Cameron Ave SW Calgary Alberta Canada T2T 0K4 admin@madisons.ca		Shipping Method: Pickup in store Ship as Complete: Yes			
PRODUCT		AVAILABILITY	QTY	EACH	TOTAL
	White Fabric Roll Arm Chaise SKU: FULO-0101	In-Stock	1	\$449.99	\$449.99
				Order Subtotal:	\$449.99
				Discount:	\$0.00
				Tax:	\$0.00
				Shipping:	\$0.00
				Shipping Tax:	\$0.00
				Order Total:	\$449.99

Figure 4-15 Order summary

Because the shopper continues the checkout as a guest shopper, the store needs the shopper's e-mail address to send an order confirmation and to inform the shopper when the merchandise will be ready to be picked up.

This page also supplies an option to send the shopper a Short Message Service (SMS) message for order confirmation. It is optional.

The online shopper enters an e-mail address and a mobile phone number.

Note: The shopper must enter a valid e-mail address here to check out as a guest shopper. For a *registered* shopper who has logged on to the store, the shopper is not prompted to input an e-mail address. The e-mail address in the registration profile is used.

5. The online shopper submits the order.

After entering an e-mail address and mobile phone number, the online shopper clicks the **Order** button to submit the order, as shown in Figure 4-16.

The screenshot shows a web form titled "Billing Information" with a yellow header. The form is divided into two main sections: "BILLING ADDRESS" and "BILLING METHOD". Under "BILLING METHOD", there is a "Pay In Store" option and a field for "E-mail address" with the value "shopper@mycom.com". Below this, the "Amount" is listed as "\$449.99". A checkbox labeled "Send SMS notifications to mobile phone" is checked. The "Country" is set to "Afghanistan" with a dropdown arrow. The "Mobile Phone Number" field contains "+93" and "1234567890". An example number "+91 1234567890" is shown below. At the bottom, there are "Back" and "Order" buttons.

Billing Information	
BILLING ADDRESS	BILLING METHOD
	Pay In Store
	* E-mail address: <input type="text" value="shopper@mycom.com"/>
	Amount: \$449.99
<input checked="" type="checkbox"/> Send SMS notifications to mobile phone	
Country: <input type="text" value="Afghanistan"/>	Mobile Phone Number: <input type="text" value="+93"/> <input type="text" value="1234567890"/>
	Example: +91 1234567890
<input type="button" value="Back"/> <input type="button" value="Order"/>	

Figure 4-16 Order submission

When the order placed successfully, the order confirmation page displays, as shown in Figure 4-17. On this page, the shopper can view the order number, the order submission date, and a summary of the order for the pickup address and billing address.

It is highly recommended that the shopper to print this page, because in-store personnel might ask for this information when the shopper picks up the merchandise in the store.

Sometime later, the store sends a confirmation e-mail to the shopper about this order. If the shopper input a mobile phone number, an SMS order confirmation message is also sent to the shopper.

When picking up the merchandise, in-store personnel might ask the shopper to show the e-mail confirmation letter or the SMS message. It depends on the merchant's choice.

Thank you for your order!

Continue Shopping

You will receive a confirmation by e-mail to verify your order.

Order number: 13501


Order date: August 13, 2009

Shipping Information

Shipping Address:
 Calgary Mall Calgary Mall
 1025 Cameron Ave SW
 Calgary Alberta
 Canada T2T 0K4
 admin@madisons.ca

Ship as Complete: Yes

Shipping Method: Pickup in store

PRODUCT	AVAILABILITY	QTY	EACH	TOTAL
 <div> White Fabric Roll Arm Chaise SKU: FULO-0101 </div>	In-Stock	1	\$449.99	\$449.99

Order Subtotal:

\$449.99

Discount:

\$0.00

Tax:

\$0.00

Shipping:

\$0.00

Shipping Tax:

\$0.00

Order Total:

\$449.99

Billing Information

BILLING ADDRESS

BILLING METHOD
 Pay In Store

 Amount:
 \$449.99

Figure 4-17 Order confirmation

4.3 The Store Locator feature

IBM WebSphere Commerce provides a simple API to enhance the Store Locator functionality. You can customize the Store Locator to integrate with map service providers, providing enhanced searching and displaying capabilities on the Store Locator page.

Integrating with map service providers provides the following enhancements:

- ▶ Store locations can be found by entering a Zip or postal code into the search fields. This advanced level of location searching enables the customer to find stores relative to any location of their choice.
- ▶ A map of the area can be displayed on the Store Locator page, where the store locations are labelled on the map. Multiple store locations labelled on the map enable the customer to quickly locate the most convenient store location.

The default Store Locator implementation generates the store list based on the city ID. You can modify the implementation to include map service provider integration, where the generated store list is based instead on the geocode latitude and longitude values.

Table 4.1 shows a comparison between default implementation in IBM WebSphere Commerce and the customization that is required to integrate with a third-party map service provide.

Table 4-1 Store Locator implementation comparison

Default implementation	Integrating with a map service provider implementation
<pre> <wcf:getData type="com.ibm.commerce.store.facade.datatype s.PhysicalStoreType[]" var="physicalStores" varException="physicalStoreException" expressionBuilder="findPhysicalStoresByGeoNo deUniqueId"> <wcf:param name="accessProfile" value="IBM_Store_Details" /> <wcf:param name="uniqueId" value="\${cityId}" /> </wcf:getData> </pre> <p>Where:</p> <ul style="list-style-type: none"> ► The uniqueId parameter generates the store list based on the cityId value. 	<pre> <wcf:getData type="com.ibm.commerce.store.facade.datatype s.PhysicalStoreType[]" var="physicalStores" varException="physicalStoreException" expressionBuilder="findPhysicalStoresFromGeo Code"> <wcf:param name="accessProfile" value="IBM_Store_Details" /> <wcf:param name="latitude" value="\${geoCodeLatitude}" /> <wcf:param name="longitude" value="\${geoCodeLongitude}" /> </wcf:getData> </pre> <p>Where:</p> <ul style="list-style-type: none"> ► A combination of the latitude and longitude parameters generates the store list based on the geoCodeLatitude and geoCodeLongitude values.

Important: You should be familiar with your map service provider's API when integrating its enhancements into the Store Locator.


4.4 MapQuest integration for Store Locator

In this section, we use MapQuest mapping services for the Store Locator functionality. MapQuest mapping service allows you to display a location map in a predefined <div> on an HTML page. The map can be centered around a specified longitude and latitude, and you can specify a default zoom level. It also allows you to add markers to display location of stores.

Important: To use MapQuest API, you must register with MapQuest and obtain a client ID, password, and API keycode. Also, you need to download the JavaScript API package from the MapQuest Developer Network at:


<http://developer.mapquest.com>

By default in the Madisons starter store, the Store Locator functionality displays only the address of the stores for a particular Country, State, and City combination, as shown in Figure 4-18.


MADISONS

[Home](#) | [Shopping Cart](#) | [Advanced Search](#) | [Store Locator](#) | [Sign Out](#)

[Furniture](#)
[Tableware](#)
[Kitchenware](#)
[Apparel](#)

 Cart: 1 item(s) subtotal: \$79.00

[Home](#) | [Store Locator](#)
US Dollar

Your Store List

Hide

STORE NAME AND ADDRESS	HOURS	
Calgary Circle Mall 300 MacLeod Tr Calgary, Alberta T2G 5R1 511.513.5789	Mon-Fri: 10am - 9pm Sat: 9am - 7pm Sun: 11am - 6pm	✕ Remove
Calgary Mall 1025 Cameron Ave SW Calgary, Alberta T2T 0K4 367.666.6666	Mon-Fri: 10am - 9pm Sat: 9am - 7pm Sun: 11am - 6pm	✕ Remove

Store Locator

 Select a location to find a store near you.

Country:
 State/Province:
 City:

Store Locator Results

 Select one or more stores to check product availability.

STORE NAME AND ADDRESS	HOURS	SELECT STORE
Calgary Circle Mall 300 MacLeod Tr Calgary, Alberta T2G 5R1 511.513.5789	Mon-Fri: 10am - 9pm Sat: 9am - 7pm Sun: 11am - 6pm	<input type="button" value="Add to store list"/>
Calgary Mall 1025 Cameron Ave SW Calgary, Alberta T2T 0K4 367.666.6666	Mon-Fri: 10am - 9pm Sat: 9am - 7pm Sun: 11am - 6pm	<input type="button" value="Add to store list"/>

Add a store location to your store list.

Figure 4-18 Default Store Locator page in Madisons starter store

Now, you can add a page section that displays the stores location in a MapQuest map as follows:

1. Extract the JavaScript files from the downloaded \clients\javascript\compactJS compressed file. Place the extracted files in the *Stores/WebContent/Madisons/javascript/MapQuest* directory.
2. Include MapQuest JavaScript scripts in *Stores/WebContent/Madisons/Snippets/StoreLocator.jsp*. Example 4-1 shows a snippet from *StoreLocator.jsp*.

Example 4-1 Include MapQuest JavaScript scripts

```
<script
src="http://btilelog.access.mapquest.com/tilelog/transaction?transac
tion=script&key=<fmt:message key="MAP_API_KEY" bundle="{storeText}"
/>&itk=true&v=5.3.s&ipkg=controls1"></script>
<!-- The JSAPI Source files -->
<script src='<c:out
value="{jsAssetsDir}">/>javascript/MapQuest/mqcommon.js' type='text/j
avascript'></script>
<script src='<c:out
value="{jsAssetsDir}">/>javascript/MapQuest/mqutils.js' type='text/ja
vascript'></script>
<script src='<c:out
value="{jsAssetsDir}">/>javascript/MapQuest/mqobjects.js' type='text/
javascript'></script>
<script src='<c:out
value="{jsAssetsDir}">/>javascript/MapQuest/mqexec.js'
type='text/javascript'></script>
```

3. Add a <div> to display the map as shown in Example 4-2.

Example 4-2 Adding a <div> area for displaying map

```
<div class="number_info" id="hideMap" style="display:block;"><a
href="Javascript:storeLocatorJS.hideMap();" class="blue"><span
class="font2"><fmt:message key="HIDE_MAP" bundle="{storeText}"
/></span></a></div>
    <div class="number_info" id="showMap" style="display:none;"><a
href="Javascript:storeLocatorJS.showMap();" class="blue"><span
class="font2"><fmt:message key="SHOW_MAP" bundle="{storeText}"
/></span></a></div>
    <br clear="all"/><br />
```

4. Create a new JavaServer Pages (JSP) `StoreLocatorMapResults.jsp` to return the geographical location of the stores in a JSON object as shown in Example 4-3.

Example 4-3 JSP rendering JSON objects containing geographical location data

```
/*-----
--
/* Licensed Materials - Property of IBM
/*
/* WebSphere Commerce
/*
/* (c) Copyright IBM Corp. 2008
/*
/* US Government Users Restricted Rights - Use, duplication or
/* disclosure restricted by GSA ADP Schedule Contract with IBM
Corp.
/*
/*-----
--
/*
%>
<%--
    *****
    * This JSP constructs the store locator map results.
    *****
--%>

<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="http://commerce.ibm.com/base" prefix="wcbase" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn"
%>
<%@ taglib uri="flow.tld" prefix="flow" %>
<%@ taglib uri="http://commerce.ibm.com/foundation" prefix="wcf" %>
<%@ include file="../../include/JSTLEnvironmentSetup.jspf" %>

<c:set var="cityId" value="-999" />
<c:if test="${!empty WCPParam.cityId}">
    <c:set var="cityId" value="${WCPParam.cityId}" />
</c:if>
<c:if test="${!empty param.cityId}">
    <c:set var="cityId" value="${param.cityId}" />
</c:if>
```



```

<c:set var="geoCodeLatitude" value="-999" />
<c:if test="${!empty WCPParam.geoCodeLatitude}">
    <c:set var="geoCodeLatitude" value="${WCPParam.geoCodeLatitude}" />
</c:if>
<c:if test="${!empty param.geoCodeLatitude}">
    <c:set var="geoCodeLatitude" value="${param.geoCodeLatitude}" />
</c:if>

<c:set var="geoCodeLongitude" value="-999" />
<c:if test="${!empty WCPParam.geoCodeLongitude}">
    <c:set var="geoCodeLongitude" value="${WCPParam.geoCodeLongitude}"
/>
</c:if>
<c:if test="${!empty param.geoCodeLongitude}">
    <c:set var="geoCodeLongitude" value="${param.geoCodeLongitude}" />
</c:if>

<c:choose>
<c:when test="${cityId != '-888'}">
    <wcf:getData
type="com.ibm.commerce.store.facade.datatypes.PhysicalStoreType[]"
    var="physicalStores"
varException="physicalStoreException"
expressionBuilder="findPhysicalStoresByGeoNodeUniqueID">
    <wcf:param name="accessProfile" value="IBM_Store_Details" />
    <wcf:param name="uniqueId" value="${cityId}" />
    </wcf:getData>
</c:when>
<c:otherwise>
    <wcf:getData
type="com.ibm.commerce.store.facade.datatypes.PhysicalStoreType[]"
    var="physicalStores"
varException="physicalStoreException"
expressionBuilder="findPhysicalStoresFromGeoCode">
    <wcf:param name="accessProfile" value="IBM_Store_Details" />
    <wcf:param name="latitude" value="${geoCodeLatitude}" />
    <wcf:param name="longitude" value="${geoCodeLongitude}" />
    </wcf:getData>
</c:otherwise>
</c:choose>

{
<c:if test="${empty physicalStoreException}">
    <c:set var="resultNum" value="${fn:length(physicalStores)}" />
    <c:if test="${resultNum == 0 && cityId == '-888'}">

```

```

        'centerLatitude':<c:out value="\${geoCodeLatitude}" />,
        'centerLongitude':<c:out value="\${geoCodeLongitude}" />,
        'resultSize':<c:out value="\${resultNum}" />
    </c:if>
    <c:if test="\${resultNum > 0}">
        <c:choose>
            <c:when test="\${cityId != '-888'}">
                'centerLatitude':<c:out
value="\${physicalStores[0].locationInfo.geoCode.latitude}" />,
                'centerLongitude':<c:out
value="\${physicalStores[0].locationInfo.geoCode.longitude}" />,
                </c:when>
                <c:otherwise>
                    'centerLatitude':<c:out value="\${geoCodeLatitude}" />,
                    'centerLongitude':<c:out value="\${geoCodeLongitude}" />,
                    </c:otherwise>
                </c:choose>
                'resultSize':<c:out value="\${resultNum}" />,
                'physicalStore':[
                    <c:forEach var="i" begin="0" end="\${resultNum-1}">
                        {'latitude':<c:out
value="\${physicalStores[i].locationInfo.geoCode.latitude}" />,
                        'longitude':<c:out
value="\${physicalStores[i].locationInfo.geoCode.longitude}" />,
                        'htmlNum':<c:out value="\${i+1}" />. ',
                        'htmlIdentifier':<c:out
value="\${physicalStores[i].physicalStoreIdentifier.externalIdentifier}" />',
                        'htmlAddress1':<c:out
value="\${physicalStores[i].locationInfo.address.addressLine[0]}"
/>',
                        'htmlAddress2':<c:out
value="\${physicalStores[i].locationInfo.address.city}" />, <c:out
value="\${physicalStores[i].locationInfo.address.stateOrProvinceName}"
/> <c:out
value="\${physicalStores[i].locationInfo.address.postalCode}" />'
                        <c:if test="\${i < resultNum -1}">
                            ,
                        </c:if>
                    </c:forEach>
                ]
            </c:if>
        </c:if>
    }
}

```

5. Create a view entry `AjaxStoreLocatorMapResultsView` in the struts configuration file for the new JSP. Update `struts-config-ext.xml` to contain the settings shown in Example 4-4. Use the appropriate *storeId*.

Example 4-4 The struts-config-ext.xml entry

```
<!-- Global Forwards -->
  <global-forwards>
    <forward className="com.ibm.commerce.struts.ECActionForward"
name="AjaxStoreLocatorMapResultsView/storeId"
path="/Snippets/StoreLocator/StoreLocatorMapResults.jsp"/>
    ...

  </global-forwards>

<!-- Action Mappings -->
  <action-mappings type="com.ibm.commerce.struts.ECActionMapping">
    <action path="/AjaxStoreLocatorMapResultsView"
type="com.ibm.commerce.struts.BaseAction">
      <set-property property="credentialsAccepted"
value="storeId:P"/>
    </action>
    ...

  </action-mappings>
```

6. Grant access for all site users to the new view `AjaxStoreLocatorMapResultsView`. Create a policy XML file `ACPforMapService.xml` as shown in Example 4-5, and place it in the `<WCDE Install Dir>/xml/policies/` directory.

Example 4-5 Sample access control policy file

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no" ?>
<!DOCTYPE Policies SYSTEM "../dtd/accesscontrolpolicies.dtd">

<Policies>

  <Action Name="AjaxStoreLocatorMapResultsView"
CommandName="AjaxStoreLocatorMapResultsView">
    </Action>

  <ActionGroup Name="MadisonsAllUsersViews"
OwnerID="RootOrganization">
    <ActionGroupAction
Name="AjaxStoreLocatorMapResultsView"/>

  </ActionGroup>

</Policies>
```

</ActionGroup>

</Policies>

7. Load the policy using the following command:
`acpload ACPforMapService.xml`
8. Check `<WCDE Install Dir>/logs/acpload.log` for any errors.

Important: If you are using a derby database, make sure the server is not running when you run the **acpload** command. If you are using other database, use the appropriate syntax for the **acpload** command.

9. Attach the properties names-values shown in Example 4-6 to the `Stores\WebContent\WEB-INF\classes\Madisons\storetext.properties` file. Use the appropriate values for the *API Keycode*.

Example 4-6 Properties used in the JSP file

```
# StoreLocator.jsp
ENTER_ZIPPOSTALCODE = Zip/Postal Code:
OR = Or
GO_BUTTON_LABEL = GO
HIDE_MAP = Hide Map
SHOW_MAP = View Map

# StoreLocator.jsp - error messages
MISSING_ZIPCODE_FIELD = This is required field. Please enter the
value.

# Do not translate begins
-----
# MAP_API_KEY to be replaced by Map API key (Google or Mapquest) of
the server
# StoreLocator.jsp, StoreLocator.js
MAP_API_KEY = <API Keycode>
# Do not translate ends
-----
```

10. Add the code shown in Example 4-7 to the following JavaScript file:

Stores/WebContent/Madisons/javascript/StoreLocatorArea/StoreLocator.js

Example 4-7 New methods and attributes for StoreLocator object

```
/* variables declarations */
resultGeocodeLatitude: null,
resultGeocodeLongitude: null,
resultCityId: null,

/**
 * This function stores the parameters that are used to generated
the result area.
 *
 * @param renderContext The render context.
 *
 */
saveResults:function(renderContext) {
    storeLocatorJS.resultGeocodeLatitude =
renderContext.properties["geoCodeLatitude"];
    storeLocatorJS.resultGeocodeLongitude =
renderContext.properties["geoCodeLongitude"];
    storeLocatorJS.resultCityId =
renderContext.properties["cityId"];
},

/**
 * This function hides the map and the "hide map" text, and shows
the "show map" text.
 *
 */
hideMap:function() {
    var hideMapDiv = dojo.byId("hideMap");
    var mapImplDiv = dojo.byId("mapImpl");
    var showMapDiv = dojo.byId("showMap");

    hideMapDiv.style.display = "none";
    mapImplDiv.style.display = "none";

    showMapDiv.style.display = "block";
},

/**
 * This function shows the map and the "hide map" text, and hides
the "show map" text.
```

```

*
*/
showMap:function() {
    var hideMapDiv = dojo.byId("hideMap");
    var mapImplDiv = dojo.byId("mapImpl");
    var showMapDiv = dojo.byId("showMap");

    hideMapDiv.style.display = "block";
    mapImplDiv.style.display = "block";

    showMapDiv.style.display = "none";

    /* display the map */
    if (storeLocatorJS.resultGeocodeLatitude == null) {
        var serviceResponse = null;
        var ioArgs = null;
        storeLocatorJS.displayMap(serviceResponse, ioArgs);
    }
    else {
        var parameters = {};
        parameters.geoCodeLatitude =
storeLocatorJS.resultGeocodeLatitude;
        parameters.geoCodeLongitude =
storeLocatorJS.resultGeocodeLongitude;
        parameters.cityId = storeLocatorJS.resultCityId;

        dojo.xhrPost({
            url: "AjaxStoreLocatorMapResultsView",
            handleAs: "json",
            content: parameters,
            service: this,
            load: storeLocatorJS.displayMap,
            error: function(errObj,ioArgs) {
                alert("Error occurs!");
            }
        });
    }
},

/**
 * This function synchronizes the display style of the map with
the "hide Map" text. So that the map is shown
 * or hidden consistently with the "hide map" / "show map" texts.
 *
 */

```

```

syncMapDisplayStyle:function() {
    var hideMapDiv = dojo.byId("hideMap");
    var mapImplDiv = dojo.byId("mapImpl");

    if (hideMapDiv.style.display == "none") {
        mapImplDiv.style.display = "none";
    }
    else {
        mapImplDiv.style.display = "block";
    }
},

/**
 * This callback function displays / refreshes the map of the
result area using the JSON object returned. The
 * map becomes blank when:
 *     1. the page is first loaded, no search has been
performed
 *     2. search by geo node (i.e. city) is performed and 0
physical store is found
 *     3. search by zip postal code is performed and 0 physical
store is found
 *     4. search is performed and nothing is returned by Map
service provider
 *
 * @param serviceResponse The service response.
 * @param ioArgs The IO arguments.
 */
displayMap:function(serviceResponse, ioArgs) {
    storeLocatorJS.syncMapDisplayStyle();

    /* display the map only when the div is visible to the user at
the moment */
    var mapImplDiv = dojo.byId("mapImpl");
    var showMap = true;
    if (mapImplDiv.style.display == "none") {
        showMap = false;
    }

    if (showMap == true) {
        if (serviceResponse != null &&
serviceResponse.centerLatitude != null &&

```

```

        serviceResponse.centerLatitude != 'undefined' &&
serviceResponse.centerLatitude != '') {
    var centerLatitude = serviceResponse.centerLatitude;
    var centerLongitude = serviceResponse.centerLongitude;
    var resultSize = serviceResponse.resultSize;

    /* the map will be shown only when at least one store
location is found */
    if (resultSize > 0) {
        var map = new MQA.TileMap(mapImplDiv);
        var glatlng = new MQLatLng(centerLatitude,
centerLongitude);
        map.setCenter(glatlng, 10);

        for (var i=0; i<resultSize; i++) {
            var title =
serviceResponse.physicalStore[i].htmlIdentifier;
            var info =
serviceResponse.physicalStore[i].htmlAddress1 + "<br/>" +
                serviceResponse.physicalStore[i].htmlAddress2;
            glatlng = new
MQLatLng(serviceResponse.physicalStore[i].latitude,
                serviceResponse.physicalStore[i].longitude);

            var poi = new MQA.Poi(glatlng);
            poi.setValue("infoTitleHTML", title);
            poi.setValue("infoContentHTML", info);
            poi.setValue("key", i);

            map.addShape(poi);
        }
        map.addControl(new MQA.LargeZoomControl(map));
        map.setSize();
    }
}
},

```

11. Modify `refreshSearchResults()` in the following JavaScript file as shown in Example 4-8:

Stores/WebContent/Madisons/javascript/StoreLocatorArea/StoreLocator.js

Example 4-8 Modification to allow for map refresh

```
refreshSearchResults:function(fromPage) {
    var performFindFlag =
PhysicalStoreCookieJS.getValueFromCookie("WC_stFind");
    var citySelectedIndex = dojo.byId("selectCity").selectedIndex;
    if (citySelectedIndex > -1) {
        var indexToUse = citySelectedIndex;
        var indexFromSavedId =
storeLocatorJS.getSavedCitySelectionIndex();
        if (indexFromSavedId != null && indexFromSavedId !=
citySelectedIndex) {
            indexToUse = indexFromSavedId;
            dojo.byId("selectCity").options[indexToUse].selected =
true;
        }

        if (performFindFlag != null) {
            if (performFindFlag == 1) {
                wc.render.updateContext('storeLocatorResultsContext',
{'cityId':dojo.byId("selectCity").options[indexToUse].value,
'geoCodeLatitude':'-888', 'geoCodeLongitude':'-888',
'fromPage':fromPage});
            }
        }
    }
},
```

12. Modify the following JavaScript file to refresh the store location on the map as shown in Example 4-9:

*Stores/WebContent/Madisons/javascript/StoreLocatorArea/StoreLocatorC
ontrollersDeclaration.js*

Example 4-9 The `postRefreshHandler()` function to refresh store location on the map

```
postRefreshHandler: function(widget) {
    var controller = this;
    var renderContext = this.renderContext;

    /* store the parameters (i.e. the state of the page) so the
map can be loaded properly */
```

```

/* for view map after hide map */
storeLocatorJS.saveResults(renderContext);

/* refresh the map on the result area */
var parameters = {};
parameters.geoCodeLatitude =
renderContext.properties["geoCodeLatitude"];
parameters.geoCodeLongitude =
renderContext.properties["geoCodeLongitude"];
parameters.cityId = renderContext.properties["cityId"];

dojo.xhrPost({
    url: "AjaxStoreLocatorMapResultsView",
    handleAs: "json",
    content: parameters,
    service: this,
    load: storeLocatorJS.displayMap,
    error: function(errObj,ioArgs) {
        alert("error");
    }
});

var bopisTable = dojo.byId("bopis_table");
if (bopisTable != null && bopisTable != "undefined") {
    bopisTable.focus();
}
var noStoreMsg = dojo.byId("no_store_message");
if (noStoreMsg != null && noStoreMsg != "undefined") {
    noStoreMsg.focus();
}

cursor_clear();
}

```

13. Start the server.

14. Open the Madisons starter store home page in a browser, and go to the Store Locator page using the link at the right-hand, top corner of the page (Figure 4-19).



Figure 4-19 Store Locator link

15. In the Store Locator page, click **Go** to the right of the City drop-down menu to display all the stores in that city (Figure 4-20).

Country:
State/Province:
City:

Canada

Alberta

Calgary

GO

Hide Map

Store Locator Results Select one or more stores to check product availability.

STORE NAME AND ADDRESS	HOURS	SELECT STORE
Calgary Circle Mall 300 MacLeod Tr Calgary, Alberta T2G 5R1 511.513.5789	Mon-Fri: 10am - 9pm Sat: 9am - 7pm Sun: 11am - 6pm	<div>Add to store list</div>
Calgary Mall 1025 Cameron Ave SW Calgary, Alberta T2T 0K4 367.666.6666	Mon-Fri: 10am - 9pm Sat: 9am - 7pm Sun: 11am - 6pm	<div>Add to store list</div>

Add a store location to your store list.

Continue Shopping

Figure 4-20 Result of store search in a city

4.5 MapQuest Geocoding

MapQuest Geocoding allows a user to obtain the geographical co-ordinates (longitude and latitude) for a given address. The input address can contain all details (such as the street, city, zip code, and country information) or just the zip code and country information.

In this section, we explain how to use this feature to obtain all the nearby stores for a zip code.

4.5.1 Install MapQuest JavaScript API proxy

Because of cross-site scripting limitations, MapQuest provides a JavaScript API which communicates with a proxy in the same domain as the requested page. The proxy in turn forwards the request to the MapQuest server. The result is then returned to the JavaScript API at the client.

In this section, we set up the JavaScript API proxy on the IBM WebSphere Commerce server as follows:

1. Extract the JSAPIProxyPage from the downloaded clients compressed file `\clients\javascript\proxy\java\JSAPIProxyPage`. The JSAPIProxyPage directory contains the files listed in Example 4-10.

Example 4-10 Content of the JSAPIProxyPage directory

```
com (directory)
README.txt
TestXML.html
Utils.js
web.xml
```

2. Open WebSphere Commerce Toolkit and create a new Dynamic Web project (Figure 4-21). Enter a Project name. Make sure that the “Add project to an EAR” box is selected. Click **Next**.

New Dynamic Web Project

Dynamic Web Project
Create a standalone Dynamic Web project or add it to a new or existing Enterprise Application.

Project name: MQAPI

Contents
☒ Use default
Directory: C:\IBM\WCDE_ENT70\workspace\MQAPI Browse...

Target Runtime
WebSphere Application Server v7.0 New...

Dynamic Web Module version
2.5

Configuration
Default Configuration for WebSphere Application Server v7.0 Modify...
A good starting point for working with WebSphere Application Server v7.0 runtime. Additional facets can later be installed to add new functionality to the project.

EAR Membership
☒ Add project to an EAR
EAR Project Name: WC New...

< Back Next > Finish Cancel

Figure 4-21 New Dynamic Web Project wizard

3. Enter a Context Root, and clear the “Generate deployment descriptor” option as shown in Figure 4-22. In this scenario, we use the deployment descriptor that MapQuest provides. Click **Finish**.

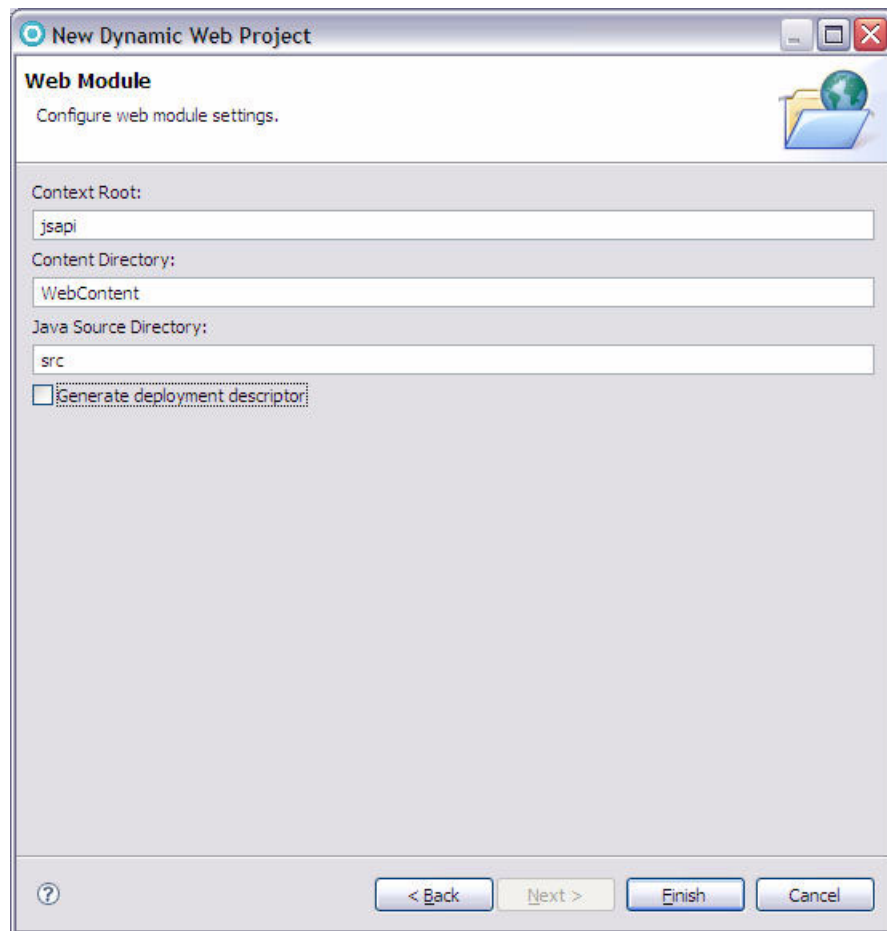


Figure 4-22 Step 2 of create Dynamic Web Project wizard

In the Java perspective, you see that a new project is created, as shown in Figure 4-23.

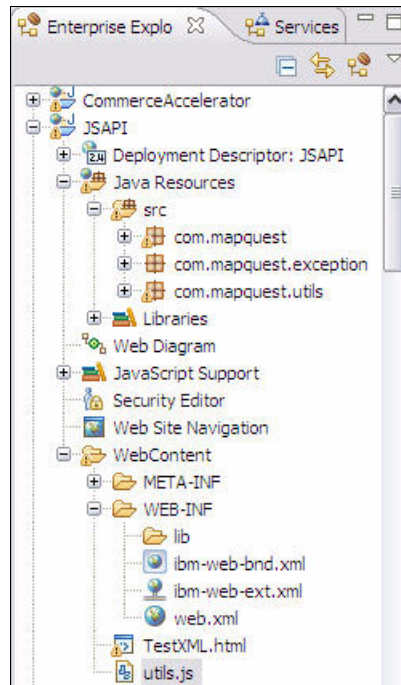



Figure 4-23 JavaScript API proxy project

4. Copy the com folder from JSAPIProxyPage to the src folder.

Important: Make sure that **Project**  **Build Automatically** is turned on. Otherwise, you will have to rebuild the project to make sure that the Java code is compiled.

5. Copy the web.xml file from JSAPIProxyPage to WEB-INF.
6. Copy the TestXML.html and utils.js files from JSAPIProxyPage to the WebContent folder.
7. Open the web.xml file. Change the values shown in Example 4-11 to the appropriate values, and save the file.

Example 4-11 Replace with your MapQuest Client ID and Password

```
<param-value>Your_ClientId</param-value>
<param-value>Your_Password</param-value>
```

8. Start the server.
9. Open the following URL in a browser:
<http://localhost/jsapi/TestXML.html>
 The MapWare XML Test Page should display.
10. Make sure *sname* is the appropriate server name in the MapQuest request URL:
[JSReqHandler?sname=geocode.dev.mapquest.com&spath=mq&sport=80](http://localhost/jsapi/TestXML.html?JSReqHandler?sname=geocode.dev.mapquest.com&spath=mq&sport=80)
11. Click **Send Data**.

The results of the default geocoding request should display at the bottom of the page as shown in Figure 4-24.

Server to send request to:

data to send:

```

<Geocode Version="1">
  <Address>
    <AdminArea1>US</AdminArea1>
    <PostalCode>80203</PostalCode>
  </Address>
  <AutoGeocodeCovSwitch>
    <Name>mqqauto</Name>
    <MaxMatches>0</MaxMatches>
  </AutoGeocodeCovSwitch>
  <Authentication Version="2">
    <TransactionInfo></TransactionInfo>
  </Authentication>
</Geocode>
    
```

response:

```

<?xml version="1.0" encoding="ISO-8859-1"?>
<GeocodeResponse><LocationCollection Count="1"><GeoAddress><AdminArea1>US</AdminArea1><AdminArea2>A
County</AdminArea4><AdminArea5>Denver</AdminArea5><PostalCode>80203</PostalCode><LatLng>
<ResultCode>ZiXAA</ResultCode><SourceId>navt</SourceId></GeoAddress></LocationCollection
    
```

Figure 4-24 A successful geocode response from MapQuest

Important: Make sure that this function is working before you continue to the next section.

4.5.2 Integration with IBM WebSphere Commerce

In the previous section, we explained how to configure the JavaScript API proxy, which communicates with MapQuest servers for Geocoding services. In this section, we describe how to use the MapQuest Geocoding API to make a geocoding request based on zip code and display the same on the map:

1. Modify the JSP for zip code entry.
 - a. Include an input box enabling a user to enter a zip code as shown in Example 4-12.

Example 4-12 Include an input box enabling user to enter a zip code

```
<input name="textZipPostalCode" id="textZipPostalCode" type="text"
class="input" height="19" size="14" onKeyPress="Javascript:if
(event.keyCode == 13) {
storeLocatorJS.preRefreshResultsFromZip(document.${formName});
return false; } else return;" />
```

- b. Include the `<script>` tag as shown in Example 4-13 to display an alert in case the user does not enter a zip code before clicking **OK** for a zip code search.

Example 4-13 Message for missing zip code

```
<script>
    MessageHelper.setMessage("MISSING_ZIPCODE_FIELD", "<fmt:message
key='MISSING_ZIPCODE_FIELD' bundle='${storeText}' />");
</script>
```

2. Add the code shown in Example 4-14 to the following JavaScript file:

Stores/WebContent/Madisons/javascript/StoreLocatorArea/StoreLocator.js

The `proxyServerName` and `proxyServerPort` strings are empty strings because the same server is acting as the proxy for MapQuest request as the server for the store pages. Use appropriate values if that is not the case in your environment.

Also use appropriate values for *ProxyServerPath*, *serverName*, and *serverPort*.

Important: While performing a zip code search, we use a default value of *Canada* for the country. Make sure that you use an appropriate country on which your site user can perform a zip code search.

Example 4-14 New methods for StoreLocator object

```
/**
 * This function manages the cookie values when OK button is
pressed from zip search.
 *
 * @param {String} cookieValue The value to be set at the cookie
key.
 *
 */
manageCookieFromZip:function(cookieValue) {
    PhysicalStoreCookieJS.setValueToCookie("WC_stZip",
cookieValue);
    PhysicalStoreCookieJS.setValueToCookie('WC_stFind', 2);
},

/**
 * This function creates a request to Map service provider
requesting the geocode of a zip/postal code when non-empty
zip/postal
 * code is entered by the user.
 *
 * @param form The form that contains the text field "zip/postal
code" entered by the user.
 *
 */
preRefreshResultsFromZip:function(form) {
    reWhiteSpace = new RegExp(/^\s+$/);

    if (form.textZipPostalCode != null &&
reWhiteSpace.test(form.textZipPostalCode.value) ||
form.textZipPostalCode.value == "") {

MessageHelper.formErrorHandleClient(form.textZipPostalCode.id,
MessageHelper.messages["MISSING_ZIPCODE_FIELD"]);
        return;
    }

    /* manage cookie values */
```

```
storeLocatorJS.manageCookieFromZip(form.textZipPostalCode.value);
```

```
var proxyServerName = "";
var proxyServerPort = "";
var ProxyServerPath = "/jsapi/JSReqHandler";
var serverName = "geocode.dev.mapquest.com";
var serverPort = "80";
var serverPath = "mq";
var geoExec = new MQExec(serverName, serverPath,
    serverPort, proxyServerName, ProxyServerPath,
    proxyServerPort );
```

```
var address = new MQAddress();
address.setPostalCode(form.textZipPostalCode.value);
address.setCountry("Canada");
```

```
var gaCollection = new MQLocationCollection("MQGeoAddress");
geoExec.geocode(address, gaCollection);
var geoAddr = gaCollection.get(0);
var geoCodeLongitude = geoAddr.getMQLatLng().getLongitude();
var geoCodeLatitude = geoAddr.getMQLatLng().getLatitude();
```

```
/* refresh the result area */
wc.render.updateContext('storeLocatorResultsContext',
{'geoCodeLatitude':geoCodeLatitude,
'geoCodeLongitude':geoCodeLongitude, 'cityId':'-888',
'fromPage':StoreLocatorContextsJS.fromPage});
},
```

3. Modify `refreshSearchResults()` in the `Stores/WebContent/Madisons/javascript/StoreLocatorArea/StoreLocator.js` file.

Example 4-15 Modification to allow for map refresh using zip code

```
refreshSearchResults:function(fromPage) {
    var performFindFlag =
PhysicalStoreCookieJS.getValueFromCookie("WC_stFind");
    var citySelectedIndex = dojo.byId("selectCity").selectedIndex;
    if (citySelectedIndex > -1) {
        var indexToUse = citySelectedIndex;
        var indexFromSavedId =
storeLocatorJS.getSavedCitySelectionIndex();
```

```

        if (indexFromSavedId != null && indexFromSavedId !=
citySelectedIndex) {
            indexToUse = indexFromSavedId;
            dojo.byId("selectCity").options[indexToUse].selected =
true;
        }

        if (performFindFlag != null) {
            if (performFindFlag == 1) {
                wc.render.updateContext('storeLocatorResultsContext',
{'cityId':dojo.byId("selectCity").options[indexToUse].value,
'geoCodeLatitude':'-888', 'geoCodeLongitude':'-888',
'fromPage':fromPage});
            }
        }
    }
}

var savedZip =
PhysicalStoreCookieJS.getValueFromCookie("WC_stZip");
if (savedZip != null) {
    dojo.byId("searchByGeoNodeForm").textZipPostalCode.value =
savedZip;

    if (performFindFlag != null) {
        if (performFindFlag == 2) {

storeLocatorJS.preRefreshResultsFromZip(dojo.byId("searchByGeoNodeFo
rm"));
        }
    }
}
},

```

4. Start the server.
5. Open the Madisons starter store home page in a browser, and go to the Store Locator page using the link at the right-hand, top corner of the page (Figure 4-25).

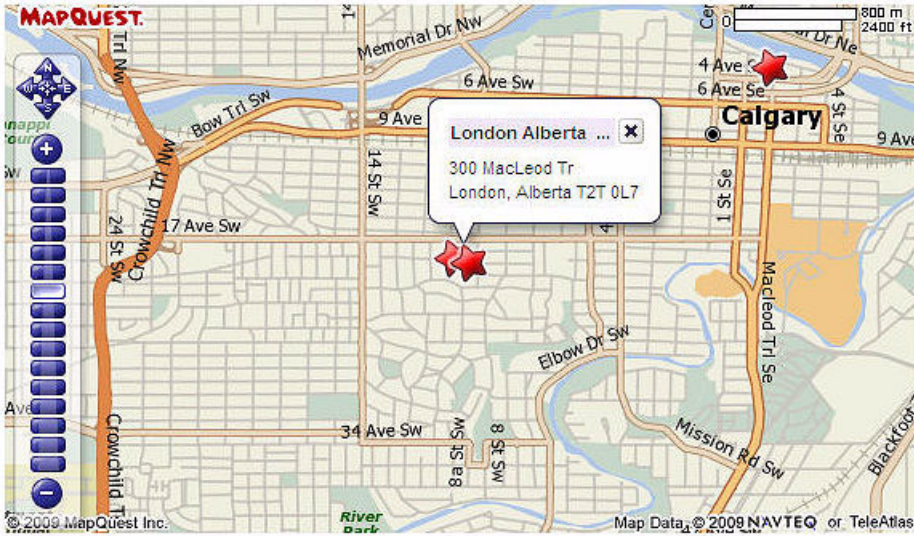


Figure 4-25 Store Locator link

6. Enter a valid Canadian zip code (we used T2T 0L7) in the Zip/Postal Code field, and click **Go**. The page displays a map section with store location pointed in the same (Figure 4-26).

Country: State/Province: City: Zip/Postal Code: Or

Hide Map



MapQuest

London Alberta ... x
300 MacLeod Tr
London, Alberta T2T 0L7

Store Locator Results Select one or more stores to check product availability.

STORE NAME AND ADDRESS	HOURS	SELECT STORE
London Alberta Mall 300 MacLeod Tr London, Alberta T2T 0L7 394.532.5789	Mon-Fri: 10am - 9pm Sat: 9am - 7pm Sun: 11am - 6pm	<input type="button" value="Add to store list"/>
Calgary Mall 1025 Cameron Ave SW Calgary, Alberta T2T 0K4 367.666.6666	Mon-Fri: 10am - 9pm Sat: 9am - 7pm Sun: 11am - 6pm	<input type="button" value="Add to store list"/>
Calgary Circle Mall 300 MacLeod Tr Calgary, Alberta T2G 5R1 511.513.5789	Mon-Fri: 10am - 9pm Sat: 9am - 7pm Sun: 11am - 6pm	<input type="button" value="Add to store list"/>

Figure 4-26 Result of a zip code search



Mobile commerce features in WebSphere Commerce V7

This chapter expands on the description of the mobile features in IBM WebSphere Commerce V7 introduced in Chapter 2, “IBM WebSphere Commerce V7 features” on page 29. It includes a more in-depth description about how you can use these mobile features and discusses specifically how to define e-Marketing Spots and promotions for mobile users.

We also describe customization scenarios that showing how to use Google Maps with the Store Locator functionality on a mobile device, how to enable product ratings and reviews for the mobile device, and how to amend the existing checkout flow in Madisons Mobile Starter Store with the ability for mobile users to have products shipped, rather than picking up the merchandise at a brick-and-mortar store.

This chapter includes the following topics:

- ▶ Madisons Mobile Starter Store
- ▶ Integration with a map service provider for the mobile Store Locator
- ▶ Buy on mobile device and select shipping address
- ▶ Product ratings and review
- ▶ Analytics

5.1 Madisons Mobile Starter Store

Today, mobile commerce is about the explosion of applications and services that have become accessible from Internet-enabled mobile devices. The devices we use are more personal, and a cross-channel solution is more relevant for most industries. WebSphere Commerce provides the best infrastructure to achieve an effective cross-channel solution in today's ever-changing mobile commerce environment.

5.1.1 Mobile commerce overview in WebSphere Commerce

Using WebSphere Commerce in mobile commerce provides the following benefits:

- ▶ Web and in-store integration
- ▶ Single view of customers and their orders
- ▶ Store Information and Locator
- ▶ Inventory visibility across channels
- ▶ Broaden customer base to mobile (in store) users

WebSphere Commerce delivers the following key features for cross-channel Mobile Commerce:

- ▶ The Madisons Mobile Starter Store template with optimized shopping flow for smartphone users that provides a common framework to reduce operating cost. Madisons mobile starter store delivers a generic mobile user interface (UI) that you can use to define the following templates:
 - Product Information and Availability
 - Order status and Tracking
 - Store and Stock Locator
 - Mobile Wish List or Shopping List
 - Mobile Marketing
- ▶ WebSphere Commerce provides Mobile Message Support, which is the WebSphere Commerce version of Short Message Service (SMS) support, for order status, promotions, and other marketing messages such as store events and sales.
- ▶ WebSphere Commerce integrates into Precision Marketing with support for mobile triggers and actions using product content and e-Marketing Spots that are targeted to mobile shoppers.

WebSphere Commerce also has the following Mobile Device Optimizations:

- ▶ Standards Compliant
 - Adheres to W3 Mobile Best Practices
 - Passes mobile OK Scheme 1.0 “Basic Tests”
- ▶ Compactness
 - Desktop accessible store pages have been reorganized
 - Some page functions omitted
 - Some images omitted
- ▶ Mobile Shopper Targeting
 - Shopper elects to receive messages to mobile device by registering phone number.
- ▶ Design
 - Easy to customize and deploy templates
 - Support mobile phones with 240*320 or higher resolution
 - Tested with Blackberry, iPhone, Nokia, Windows® Mobile, and Android
 - Partner’s transcoding services with mobile business rules
 - A common commerce engine with the following mobile business rules:
 - Device or browser detection and routing
 - Mobile specific product content and marketing/promotion
 - Channel aware mobile transactions for analytics

5.1.2 Madisons Mobile Starter Store

The Madisons Mobile Starter Store enables customers to browse the storefront using their mobile devices. Table 5-1 provides the URLs for the Madisons Mobile Starter Store and the Madisons mobile starter store.

Table 5-1 The URLs for the Madisons Starter Store and the Madisons Mobile Starter Store

Store	URL
Madisons Starter Store Web	http://localhost/webapp/wcs/stores/Madisons/index.jsp
Madisons Mobile Starter Store	http://localhost/webapp/wcs/stores/Madisons/mobile/index.jsp

The Madisons Mobile Starter Store can act as a base for implementing mobile storefronts. Deploying the Madisons Mobile Starter Store and integrating SMS support ensures that your WebSphere Commerce implementation captures the current Web and mobile technologies.

Figure 5-1 shows the Madisons Starter Store and the Madisons Mobile Starter Store Web interface.

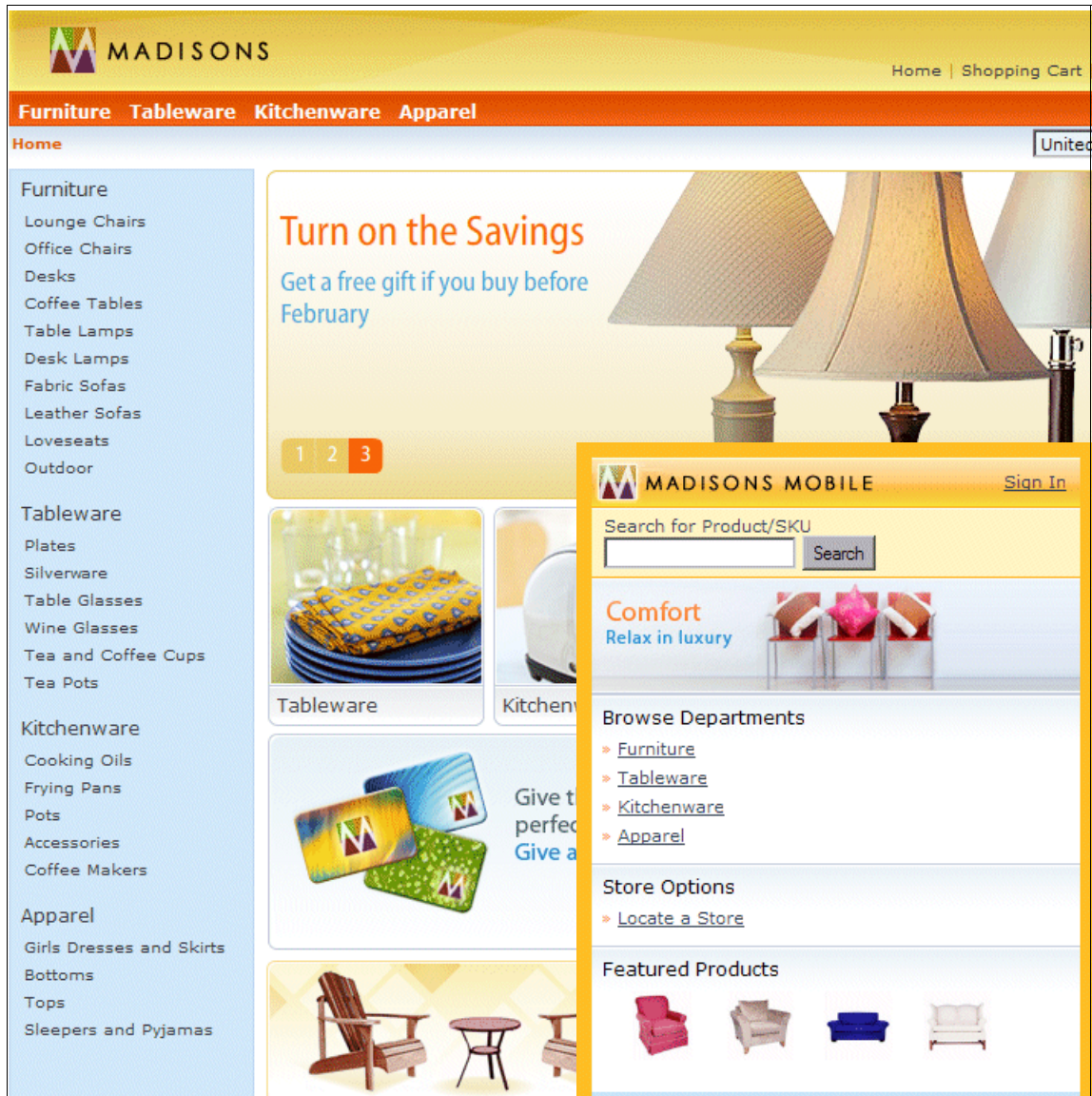


Figure 5-1 The Madisons Starter Store and the Madisons Mobile Starter Store Web interface

Main features

The Madisons Mobile Starter Store includes the following mobile optimized store pages:

- ▶ User Registration
- ▶ Mobile Marketing / Promotion
- ▶ Product Information / Availability
- ▶ Store / Stock Locator
- ▶ Mobile Shopping List (Wish List)
- ▶ Buy online, pickup in store
- ▶ Order Status and Tracking

Site flow and interactions

The Madisons Mobile Starter Store provides a comprehensive layout of components that you must identify in order to develop a mobile strategy with business staff. In this section, we analyze the Site Flow for each store page as well as the how each store page interacts with other pages.

There are four main subsystems on the Site Flow:

- ▶ Member Subsystem
Provides the pages to register a new customer and to log on with credentials. It also provides a full overview of the main assets, such as Orders, Wish List, and Suscriptions.
- ▶ Catalog Subsystem
Provides the pages for Category, Product and Searching Tasks.
- ▶ Order Subsystem:
Provides the pages for Order Information, Order Summary and Shopping Cart.
- ▶ Main Subsystem (Home Page):
Provides access to Preferences, Contact Us, Privacy Policy, Store Locator, Wish List, and Order Status.

Madisons Mobile Starter Store components

Each mobile Web page includes the following elements:

- ▶ JavaServer Pages (JSP) files, which include a JSP file for the entire page, header, footer, and recommended e-Marketing Spot JSP if applicable.
- ▶ JSP fragments, which according to checkout flow include JSP components for the product or SKU search function, department browser listing, and store options display (such as Store Locator).

- ▶ E-Marketing and Content spots, which are specialized JSPs that provide marketing information. The Madisons Mobile Starter Store e-Marketing Spots contain the prefix *Mobile* to distinguish them from Madisons Mobile Starter Store e-Marketing Spots of the same name. The mobile e-Marketing Spots include:
 - MobileApparelFeaturedProducts
 - MobileFurnitureFeaturedProducts
 - MobileHomePage
 - MobileHomePageFeaturedProducts
 - MobileKitchenwareFeaturedProducts
 - MobileTablewareFeaturedProducts
- ▶ Links and buttons, which are the reference points that navigate you to a whole document or to a specific element within a document. Links and buttons provide each mobile Web page with a more efficient user interaction model.

The next section provides an example mobile home page and discusses the components of this page.

Example mobile store pages

Figure 5-2 show a sample of mobile store pages.



Figure 5-2 Madisons Mobile Starter Store home page

The mobile home page helps customers navigate the store by displaying a search entry, followed by a list of available departments to browse, each of which contains products. The recommended category and featured products e-Marketing Spots contain highly visible product image placements, as they are directly featured on the home page of the store. Additional features of the home

page include the ability to locate a store by providing a link to the Store Locator page.

The main components of the mobile home page include JSP files, JSP file fragments, e-Marketing and Content spots, and links and buttons:

- ▶ JSP files
 - The MobileHome.jsp file represents the entire page.
 - The CachedHeaderDisplay.jsp file displays the store's header.
 - The CachedFooterDisplay.jsp file displays the store's footer.
 - The ContentAreaESpot.jsp file displays the recommended category e-Marketing Spot.
 - The FeaturedProductsESpot.jsp file displays the featured products e-Marketing Spot.
- ▶ JSP file fragments
 - The searchHeader.jspf file displays the product/SKU search function.
 - The BrowseDepartments.jspf file displays the department browsing listing.
 - The storeOptions.jspf file displays the store options (for example, the Store Locator).
- ▶ E-Marketing and Content spots:
 - The recommended category e-Marketing Spot displays.
 - The featured products e-Marketing Spot displays.
- ▶ Links and buttons
 - When the user clicks **Go**, the CatalogSearchResultsDisplayView is called.
 - When the user clicks **More info**, the recommended category from the e-Marketing Spot is called.
 - When the user clicks a department, the CategoriesDisplay is called.
 - When the user clicks **Locate a Store**, the StoreLocator is called.
 - When the user clicks a product, the ProductDisplay is called.

You can find more information about Madisons Mobile Starter Store components in the WebSphere Commerce Version 7 Information Center at:

<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.madisons-starterstore.doc/concepts/csmmadisonintro.htm>

Because Madisons Mobile Starter Store provides mobile Web page components, you can study the following main mobile flows to analyze how your company can integrate a marketing strategy:

- ▶ Mobile catalog browsing flow
- ▶ Mobile Store Locator flow
- ▶ Mobile checkout flow

Mobile catalog browsing flow

The catalog browsing flows in the Madisons Mobile Starter Store include the overall mobile layout, home, department, category, product details, product compare, advanced search, and search results pages. The following key points of this flow identify customer interaction:

- ▶ **Product Navigation and Search:** Customers can navigate by categories, products, and items. They also can search by SKU, attributes, price, range, and so forth.
- ▶ **Demand Generation:** Using these Web pages, you can add e-Marketing Spots. You can also use the Wish List as a main tool for marketing.
- ▶ **Product comparison:** Customers can make side-by-side comparison by attributes and can list inventory availability by stores and online.

Table 5-2 describes the mobile catalog browsing flow.

Table 5-2 Mobile catalog browsing flow

Pages	Description
Mobile overall layout	The overall layout remains consistent throughout the Madisons Mobile Starter Store and contains three main sections: <ul style="list-style-type: none"> ▶ The header ▶ Breadcrumb trail ▶ Footer
Mobile home	The mobile home page helps customers navigate the store by displaying a search entry, followed by a list of available departments to browse, each of which contains products. The recommended category and featured products e-Marketing Spots contain highly visible product image placements, as they are directly featured on the home page of the store. Additional features of the home page include the ability to locate a store by providing a link to the Store Locator page.
Mobile category	The mobile category pages help customers navigate through the store by only displaying products based on the category that is viewed. The level of precision is increased by viewing products and subcategories based on the category to which they belong. Products display with a picture, name, price, and stock availability.

Pages	Description
Mobile product details	The mobile product details page is the main page where the customer can view single products. Products display with a name, picture, price, stock availability, and description. The product picture is larger on the product details page than it is on other store pages, enabling the customer to observe the product in more detail. The customer can add the product to the shopping cart or wish list, select a product to compare, check the product availability, and select the previous and next product in the category.
Mobile product compare	The mobile product compare page enables the customer to compare up to four products at the same time. The products display vertically with pictures, descriptions, stock availability, and product-specific attributes in an organized table. The product compare enables customers to compare products easily without navigating to each product page and comparing the features of each product manually. A customer can remove a product directly from the compare list without leaving the product compare page.
Mobile advanced search	The mobile advanced search page enables customers to produce precise search results using several attributes. The search filters include the search term and methods for searching within the term, locations to search within, brands, price range, category, and number of results per page.
Mobile search results	The mobile search results page displays after the customer submits a search.

Figure 5-3 illustrates sample mobile store pages for the mobile catalog browsing flow.

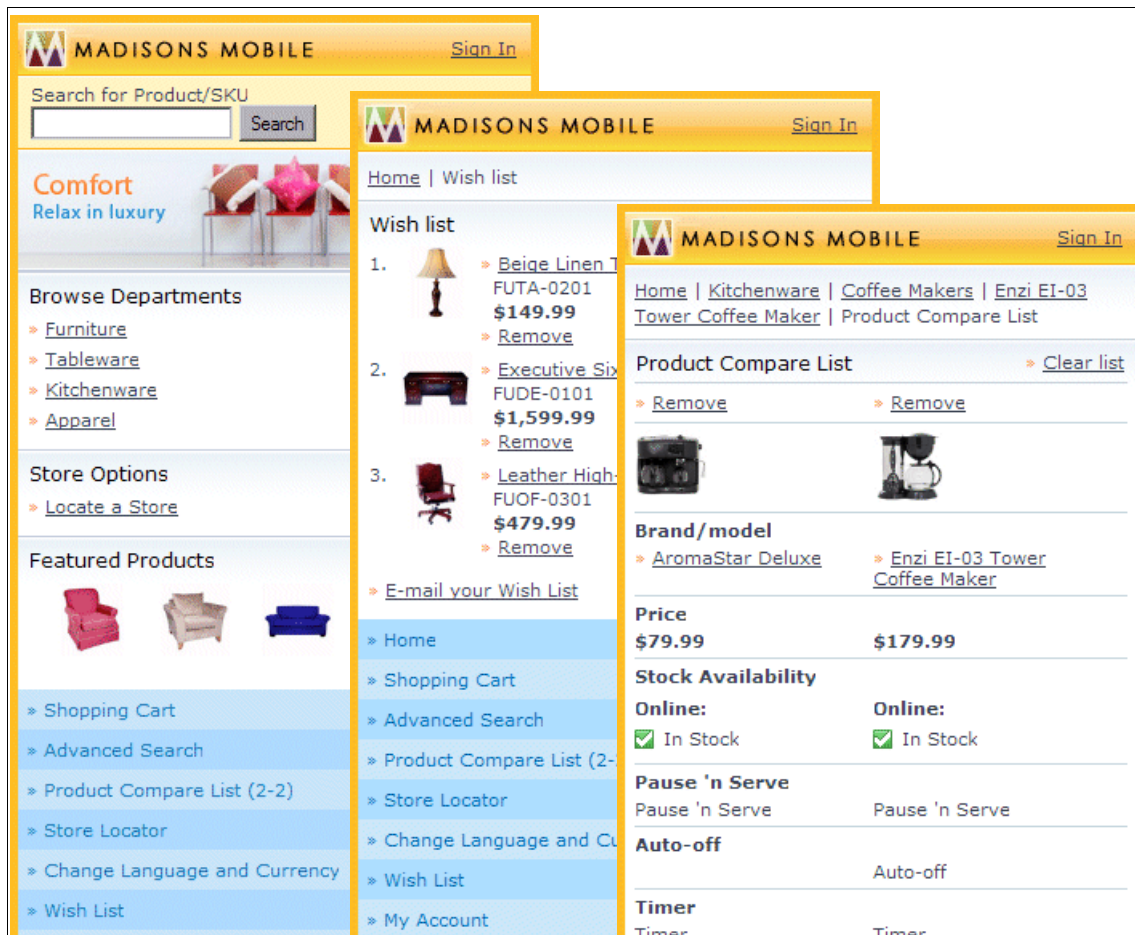


Figure 5-3 Mobile catalog browsing flow pages

Mobile Store Locator flow

The Store Locator flows in the Madisons Mobile Starter Store include the Store Locator, Store Locator results, store details, and map service provider integration scenarios. The following key points of this flow identify customer interaction:

- ▶ Store location search by city or zip code: Directions from a shopper's GPS location to the selected store.
- ▶ GPS-based store search: Show the nearest stores based on the shopper's GPS location.
- ▶ Store Information: Store hours, address, and other information or click to call the store associate.

Table 5-3 describes the mobile store locator flow.

Table 5-3 Mobile store locator flow

Pages	Description
Mobile Store Locator	Customers can use the mobile Store Locator feature to locate a nearby store. A list of stores is given based on the inputted location. The customer can customize the resulting store list to display a set number of stores. Customers can select multiple store locations. When checking the availability of a product, however, the Store Locator must be used before the availability of a product can be verified.
Mobile Store Locator results	The mobile Store Locator results displays the store's name, address, hours, and phone number, as well as the option to map the location and the option to add the store to the store list. If a partial search is made for a city name and multiple cities are found, a list displays to select the city.
Mobile Store Locator store details	The mobile store details page displays the store's name, address, and phone number, as well as the option to map the location and the store's hours.
Integrating with map service providers	The customer can customize the mobile Store Locator to integrate with map service providers, providing enhanced searching and displaying capabilities on the Store Locator page.

Figure 5-4 illustrates a sample mobile store page for the store locator browsing flow.



Figure 5-4 Store Locator mobile page

Mobile checkout standard flow

The checkout flows in the Madisons Mobile Starter Store include the Shopping Cart, sign in, Store Locator, Store Locator results, store list, billing address selection or entry, payment method, order summary, and order confirmation pages. The following key points of this flow identify customer interaction:

- ▶ Customer has access to the following product details:
 - Product image, description, and price
 - Inventory availability across channels
 - List of favorite stores by user preference
- ▶ Buy on line and pickup in store
 - Quick checkout flow
 - Payment plug-in architecture to enable mobile specific payment methods

Table 5-4 describes the Madisons Mobile Starter Store checkout standard flow.

Table 5-4 Madisons Mobile Starter Store checkout standard flow

Pages	Description
Mobile shopping cart	The shopping cart displays the items that are added to the Shopping Cart.
Mobile checkout sign in	Registered customers can use the mobile checkout sign in page to log on to the store before checkout. Signing in enables the customer to use the quick checkout, because much of their personal information is already registered with the store.
Mobile checkout Store Locator	The Store Locator is used during the mobile checkout to locate a convenient store location to pick up the order. Customers can search by zip code or city name.
Mobile checkout Store Locator results	A list of store locations displays during the mobile checkout, with each store's name, address, hours of operation, phone number, and map link. Customers can add stores to the store list. Customers must add a store to the store list to continue checkout, where they must select a store location as the pickup location.
Mobile checkout store list	The customer can select a store location from the store list to pickup the order. Each store location in the store list contains the store's name, address, hours of operation, phone number, and stock status. Customers can remove a store location from the store list. After selecting a store, the customer can continue checkout and complete billing and payment information.
Mobile checkout select billing address	The customer can select billing address information to continue with the mobile checkout. If a different billing address is required, clicking Create new address enables the customer to enter a new address during checkout.
Mobile checkout enter billing address	The customer can enter billing address information to continue with the mobile checkout. The billing fields include the recipient, first name, last name, street address, city, country or region, state or province, zip code, phone number, and e-mail address.
Mobile checkout payment method	Customers can use the mobile checkout payment page to enter payment method and information and, optionally, to enter a promotion code to receive further order discounts. The customer must click Update Order Total to apply the promotion code to the order.

Pages	Description
Mobile checkout order summary	The mobile checkout order summary displays the details submitted during the mobile checkout. The information includes the store pickup location, billing address and method, and items in the order. The order's subtotal, any discounts, tax, shipping, and total are tallied and displayed following the order information. The customer must click Place your order to complete the mobile checkout.
Mobile checkout order confirmation	The mobile checkout order confirmation displays the final details for the mobile checkout. The information includes the order number and date, store pickup location, billing address and method, and items in the order. The order's subtotal, any discounts, tax, shipping, and total are tallied and displayed following the order information. To return to the home page, the customer can click Continue shopping .

For more details about the standard checkout flow, refer to 5.4.1, “Standard checkout page flow for Madisons Mobile Starter Store” on page 221.

5.1.3 Mobile marketing

With previous versions of the Madisons mobile starter store, it was possible to add data from a marketing activity, such as a campaign. With this version of the Madisons mobile starter store, the concept of *mobile marketing* opens up a new communications channel. You can now create marketing activities that send text messages (that is, an *SMS*) to customers. For example, you can send a text message to customers to provide them with a promotion code to use when they visit your Web site or to inform them about an upcoming sale. This new feature makes it easy to integrate mobile technologies into your marketing strategy. In addition, this version of the Madisons mobile store introduces the concept of e-Marketing Spots.

Mobile marketing offers a direct, personal and time-sensitive means to get marketing messages out to your customers. You can create *Dialog activities* that capitalize on these advantages. The Dialog activity uses the Send Message action to send a text message (that is, an SMS). You can either send a text SMS message to a customer in response to a specific action (such as the customer registering with the store) or send the text message in bulk to one or more customer segments on a specific day. Either way, you can reach customers wherever they are and draw them back to your store.

Figure 5-5 shows the main page, the My Subscriptions page, to manage the customer's interaction.

MADISON'S MOBILE [Sign Out](#)

[Home](#) | [My Account](#) | [My Subscriptions](#)

My Subscriptions

Sign up to receive exclusive offers. You may stop this subscription service at any time by deselecting the checkbox options.

E-mails

☒ Send me e-mails about store specials

Texting

Mobile phone number

Country
Canada

Mobile phone number
+1 6476281234

☒ Send SMS Notifications to mobile phone
☒ Send SMS promotions to mobile phone

[» Home](#)
[» Shopping Cart](#)

Figure 5-5 New user registration and subscriptions to SMS messages mobile page

Clicking the **My Account** link opens the My Account mobile Web page. This page provides the following useful features that you can use for mobile marketing:

- ▶ **My subscriptions:** Useful for promotion messages or order notification messages.
- ▶ **My personal information:** Useful for personal profile update, payment registration for quick checkout, and for language and currency selection.
- ▶ **My orders:** Useful for order status and history.

Mobile marketing features such as SMS and e-Marketing Spots can help you develop Web activities that implement and support your marketing strategy.

Using SMS functionality for mobile marketing

IBM WebSphere Commerce V7 provides a J2EE Connector Architecture (JCA) adapter in the WebSphere Commerce messaging subsystem for SMS enablement and functionality.

WebSphere Commerce provides the following key assets for SMS solutions:

- ▶ User registration and account profile
 - Communication preference (SMS versus e-mail)
 - Opt-in and opt-out by notification types
- ▶ Real-time marketing and notification
 - Marketing campaign through SMS
 - Promotion code for qualified shoppers
 - Order confirmation and fulfillment status
 - Text for store pickup
- ▶ SMS adapter implementation
 - Hypertext Transfer Protocol (HTTP) and Parlay X interface support
 - SMS aggregator or Telecom Server

Figure 5-6 shows the logical model for SMS using WebSphere Commerce.



Figure 5-6 WebSphere Commerce for mobile marketing

As part of your mobile marketing strategy you must create marketing content for SMS using the WebSphere Commerce Management Center. You can create the content of an SMS in a Dialog activity that uses the Send Message action to send a text message to a customer. Figure 5-7 shows the message creation.

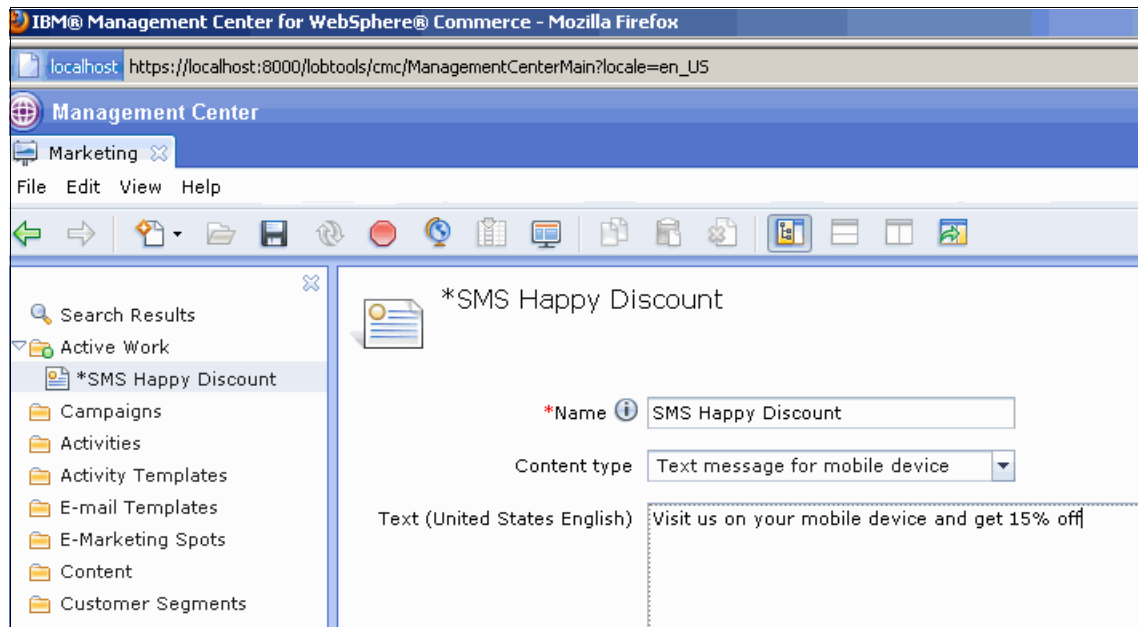



Figure 5-7 SMS Message for marketing purposes

Enabling the SMS *transport* extends cross channel access by offering mobile device capabilities in WebSphere Commerce. You enable SMS transport using the following steps:

1. Add an SMS transport to the site using the Administration Console transports.
2. Create and configure a message type using the Administration Console message types.
3. Map the message type to a JSP file by editing the `struts-config.xml` configuration file.
4. Test the new configuration to ensure that it works for your company.

Step 1: Add an SMS transport

To configure SMS transport:

1. Open the Administration Console, and select **Site or Store** on the Administration Console Site or Store selection page.
2. Click **Configuration**  **Transports**.

3. Click **Add** to open the Add Transport page.
4. Select the SMS transport that you want to add to the site. Then, click **Add** to accept the changes. When you add a transport method to a site, it is activated automatically.
5. Select the SMS transport that you created, and click **Configure**.
6. Complete the values for the parameters that correspond to the SMS transport that you created. Click **OK**.

Figure 5-8 shows a sample of available transports in WebSphere Commerce. This image shows an SMS transport addition.

Add Transport			Page 1
9 items			
<input type="checkbox"/>	Available Transports	Description	
<input type="checkbox"/>	E-mail	E-mail sender	
<input type="checkbox"/>	File	File Writer	
<input type="checkbox"/>	WebSphere MQ	WebSphere MQ	
<input type="checkbox"/>	Sample adapter	Sample adapter	
<input checked="" type="checkbox"/>	SMSHTTP	SMSHTTPDesc	
<input type="checkbox"/>	SMSWS	SMSWSDesc	
<input type="checkbox"/>	HTTP	HTTP	
<input type="checkbox"/>	WebServices(HTTP)	WebServices over HTTP	
<input type="checkbox"/>	WebServices(JMS)	WebServices over JMS	


Figure 5-8 Enabling SMS transport

Step 2: Create and configure a message type

To create a message type:

1. Open the Administration Console and choose of the following options:
 - **Site** to assign a message type to a site
 - **Store** to assign a message type to a store

For our example, we selected **Site**.

2. Click **Configuration**  **Message Types**. The Message Type Configuration page opens. Select the message type to which you want to assign a transport, and click **Change**.

In our example, we selected “Message for a received order.”

If the message type is not in the list, click **New**. The Message Transport Assignment page opens. If this is a new transport assignment, select the message type to which a transport is to be assigned from the Message Type list.

3. Enter the transport configuration values in the appropriate fields. We selected the defaults for message severity range and selected SMS messages for Device format.

Figure 5-9 shows a new message transport example.

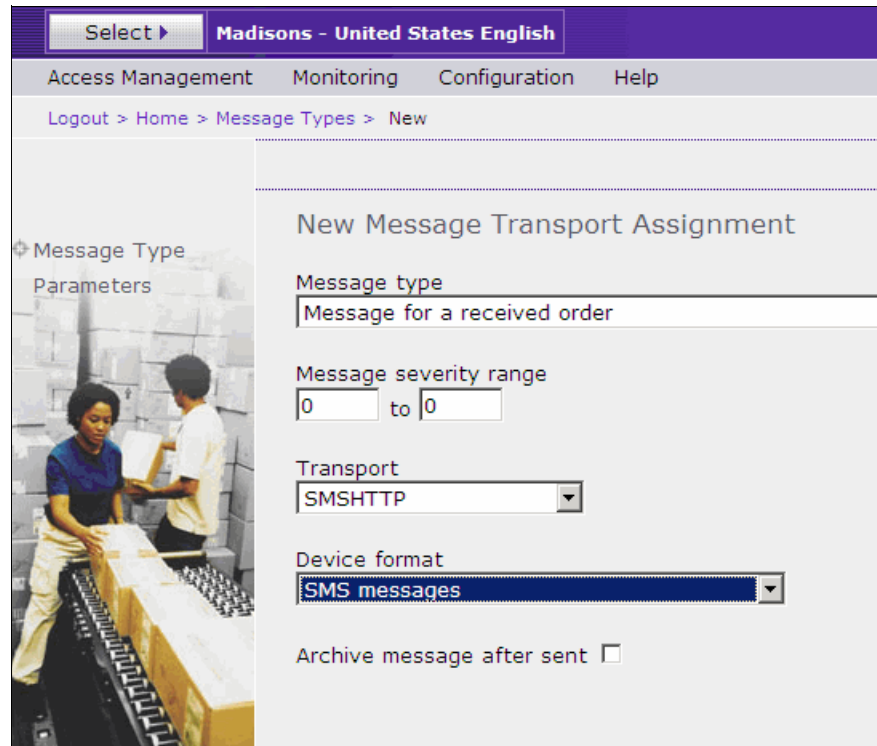


Figure 5-9 Message type configuration

4. Click **Next** to configure the transport parameters for the specified message type. For our example, we accepted the defaults.
5. Click **Finish**. The Message Type Configuration page displays, and the Transport Status Column should be Active. If the transport status is not active, the transport has been deactivated or removed.

Step 3: Mapping the message

To modify the mapping of the message template to another JSP file, you can modify the Struts configuration file as follows:

1. Open the `struts-config.xml` configuration file that is located in the `WEB-INF/stores/yourStoreName/data/` directory and add a new entry for the SMS message type that you created, as shown in Example 5-1.

Example 5-1 Optional step sample

```
<forward
className="com.ibm.commerce.struts.ECActionForwardname="yourView/0/-7"
path="/yourJSPfile.jsp">
    <set-property property="resourceClassName"
value="com.ibm.commerce.messaging.viewcommands.MessagingViewCommandImpl
"/>
<set-property property="properties" value="storeDir=no"/> <set-property
property="interfaceName"
value="com.ibm.commerce.messaging.viewcommands.MessagingViewCommand"/>
<set-property property="implClassName"
value="com.ibm.commerce.messaging.viewcommands.MessagingViewCommandImpl
"/>
<set-property property="direct" value="true"/>
</forward>
```

The configuration file uses -7 as the SMS device format, as defined in the `DEVICEFMT` database table. The path parameter in your new entry is mapped to a JSP file.

2. Save your changes and close the configuration file.
3. Restart the WebSphere Commerce server to apply your changes.

Table 5-5 lists the SMS message templates that are implemented in JSP files in the Stores-Web Content directory.

Table 5-5 SMS message templates

Message name	Template
OrderAuthorized	OrderAuthorizedSMS.jsp
OrderReceived	OrderReceivedSMS.jsp
OrderCancel	OrderCanceledNotificationSMS.jsp
OrderChanged	OrderChangedNotificationSMS.jsp
ReleaseShipNotify	ReleaseShipNotifySMS.jsp

Step 4: Testing the SMS methodology

The last step to SMS configuration is to test scenarios that are appropriate for your company. Testing reception of SMS messages requires the following components:

- ▶ SMS gateway provider
- ▶ Real Mobile Device

Note: Mobile device simulators are not capable of receiving SMS messages.

You can test message over Web service transports using the WebSphere Telecom Toolkit, which can simulate a Parlay X interface or an SMS gateway.

Using e-Marketing Spots

You can use *e-Marketing Spots*, which display marketing information to customers, to reserve space on mobile store pages. E-Marketing spots can display the following types of marketing information about store pages:

- ▶ Content, such as advertisements for promotions
- ▶ Category recommendations
- ▶ Catalog entry recommendations, including merchandising associations

Using *Web activities*, you can control the information that displays in each e-Marketing Spot. Web activities can target different types of customers with different advertisements or recommendations dynamically, so that each customer viewing the e-Marketing Spot might not see the same thing. You do, however, need to manage any non-technical limitations because you are using a mobile device, such as using merchandising association.

You can now create, change, and delete e-Marketing Spots using the Marketing tool in the Management Center. In addition, you can view impression and click statistics for each e-Marketing Spot that you use in a Web activity to evaluate the success of your marketing efforts. You can specify catalog entries, categories, and content as the default content for an e-Marketing Spot.

You name e-Marketing Spots descriptively so that the name includes the location and purpose, for example, `HomePageRow1Ads` or `CheckOutPageRecommendation`. This unique naming helps to reduce confusion about on which page the e-Marketing Spot is located and the information that it displays. If necessary, you can add numbers to the name to differentiate between two e-Marketing Spots that display on the same page.

Figure 5-10 lists examples of e-Marketing Spot names. You can use these spots to add customized options for e-Marketing Spot experiments.







E-Marketing Spots - List		
* Type	* Name	Description
	MobileApparelFeaturedProducts	Display suggestive sellings in the apparel top category mobile page
	MobileFurnitureFeaturedProducts	Display suggestive sellings in the furniture top category mobile page
	MobileHomePage	Display category recommendation in the mobile home page.
	MobileHomePageFeaturedProducts	Display suggestive sellings in the mobile home page.
	MobileKitchenwareFeaturedProducts	Display suggestive sellings in the kitchenware top category mobile page
	MobileTablewareFeaturedProducts	Display suggestive sellings in the tableware top category mobile page

Figure 5-10 *MobileHomePageFeaturedProducts e-Marketing Spot*

The move to the Management Center introduces several changes in behavior in e-Marketing Spots. If you used WebSphere Commerce Accelerator in the past to manage e-Marketing Spots, review the following information center topic to understand the differences:

<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp>

Planning and implementing e-Marketing Spots

When planning the purpose and location of e-Marketing Spots, the following people are typically involved:

- ▶ The store developer who creates the JSP files for the store pages
- ▶ The marketing manager who markets activities that use the e-Marketing Spots
- ▶ The media designer who creates any graphics or text that displays in the e-Marketing Spots

Collaboration ensures that the e-Marketing Spots are implemented in a way that provides adequate space and that retains the aesthetics of the mobile design.

To include an e-Marketing Spot on a store page, the store developer created a JSP file for the e-Marketing Spot. This JSP file specifies the following information:

- ▶ The name of the e-Marketing Spot
- ▶ The types of marketing information to display (that is, content, category recommendations, catalog entry recommendations, or a combination of these types)
- ▶ The maximum number of each type of marketing information that the e-Marketing Spot can display at one time

If the e-Marketing Spot snippet is generic, then these details are passed in as parameters to the JSP file for the store page, and they are not in the JSP file for the e-Marketing Spot.

In the Management Center, you create the e-Marketing Spot using the Marketing tool to add the e-Marketing Spot to the marketing database so that you can specify it in Web activities to display marketing messages.

Depending on the type of Web activity that you create, you might also need the following elements for your store:

- ▶ Customer Segments
- ▶ Promotions
- ▶ Content

Creating e-Marketing Spot experiments

Marketing *experiments* ensure that your brand and promotional messages are as effective as possible. Marketing experiments enable marketing managers to run alternative paths within existing Web activities to determine whether small changes might improve the effectiveness of a Web activity.

An experiment can include an experimental version of any marketing element, such as a target customer segment, an e-Marketing Spot that delivers the marketing message, or the marketing content. You create experiments by editing existing Web activities.

By changing a single element on an experimental branch, you can experiment with the following elements:

- ▶ The content that the Web activity displays
Content experiments display different messages to customers to determine which message is more effective at driving sales. These messages can include marketing content, category recommendations, product recommendations, or merchandising associations.

- ▶ The e-Marketing Spot in which the Web activity displays

An e-Marketing Spot experiment displays the same marketing message in two or more different e-Marketing Spots. You use e-Marketing Spot experiments to determine the best location on the mobile store pages for your marketing message. For example, you can display the same advertisement for patio furniture on the mobile home page and on the mobile Shopping Cart page to see which location gets better results.
- ▶ The target customer segment

Segment experiments display marketing content to a variety of customer segments to determine which segment is a better target for the message.
- ▶ The Web activity that displays in a particular e-Marketing Spot

Web activity experiments display different marketing activities, including content that targets different customer segments, in the same e-Marketing Spot to determine which Web activity is more effective at driving sales.

5.2 Promotions

Promotions enable you to offer customers incentives to purchase. WebSphere Commerce supports numerous types of promotions, such as price promotions, merchandise specials such as gifts with purchase and buy-one-get-one, and service promotions including reduced shipping costs.

You can use the Promotions tool in the Management Center to create and manage promotions that support your site's marketing campaigns.

With a cross-channel solution, your company must also provide a seamless shopping experience to shoppers. So, with mobile commerce promotions, you need the same business logic that offers incentives to customers when certain conditions are satisfied on other channels, such as Web, CSR, Point-of-Sale (POS), and so forth.

The central promotion engine evaluates promotions for customers while they shop on any channel, including mobile, so you can offer customers price incentives to drive sales, or promote a product.

The evaluation process has several stages and involves a variety of components of the promotion engine.

Figure 5-11 shows a promotion sample created in the Management Center.

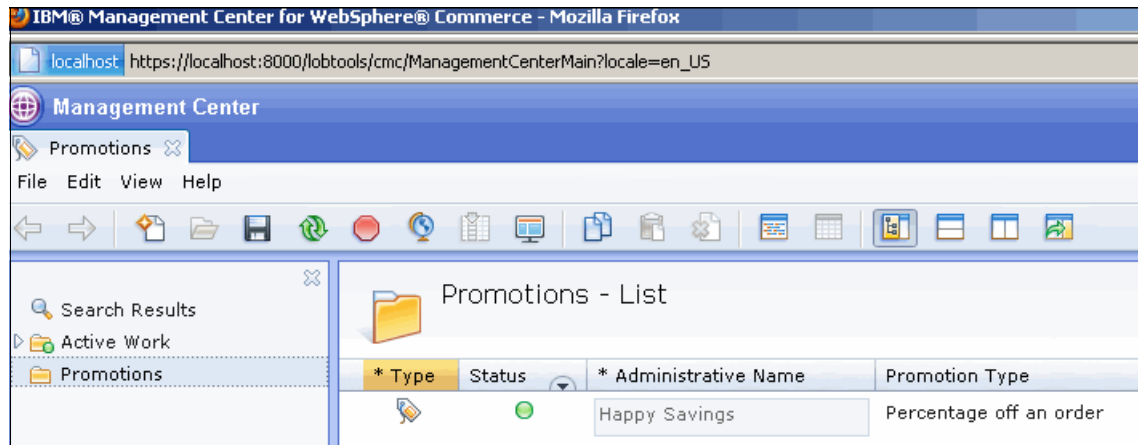


Figure 5-11 Using a promotion for mobile sample

This sample offers customers the following order-level promotion:


- ▶ Customers receive 10% off orders of \$100 or more. Figure 5-12 show these parameters.

The screenshot shows the "Happy Savings (Read-Only)" configuration page. It has three tabs: "Manage Promotion", "Descriptions", and "References". The "Manage Promotion" tab is active. Below the tabs is a section titled "Promotion Properties" with the following fields:

*Administrative name	Happy Savings
Promotion type	Percentage off an order
Redemption method	Qualifying purchase
Combination with other promotions	Combine with other promotions
*Priority	900

Figure 5-12 Discount configuration

- The promotion is available only during the first half of July. Figure 5-13 displays a schedule for this sample.

 Happy Savings (Read-Only)

Manage Promotion

Descriptions

References

▶ Promotion Properties


▶ Purchase Condition and Reward

▶ Redemption Limits


▼ Schedule

Start date ⓘ

2009/08/03




12:00 AM




Eastern Time

End date ⓘ

2009/08/14



12:00 AM



Eastern Time

Days promotion is available ⓘ

☒ Every day of the week

☐ Selected days of the week

Time of day promotion is available ⓘ

☒ All day

☐ During a specified time

Figure 5-13 Promotion schedule

- This promotion is available only to registered customers. Figure 5-14 displays how the promotion is configured to use a special customer segment.

Happy Savings (Read-Only) Save and Close

Manage Promotion Descriptions References

► **Promotion Properties**




► **Purchase Condition and Reward**


► **Redemption Limits**

► **Schedule**

▼ **Target Customer Segment**

Customer segments

Find and Add   

* Type	* Name	Description
	Registered Customers	Customers who have registered with the store

1 to 1 of 1

Figure 5-14 Customer selection

- This promotion has a limit of one redemption per customer, as shown in Figure 5-15.

Happy Savings (Read-Only)

Manage Promotion Descriptions References

► **Promotion Properties**

► **Purchase Condition and Reward**

▼ **Redemption Limits**

Maximum redemptions for this promotion ☐ Unlimited ☒ Set maximum redemption

Maximum redemptions on a single order ☐ Unlimited ☒ Set maximum redemption

Maximum redemptions by a single customer ☐ Unlimited ☒ Set maximum redemption

*Maximum redemptions 1

Figure 5-15 Promotion conditions

Using this sample, the customer uses a mobile device to make an online shopping purchase for \$249. Figure 5-16 shows the mobile checkout page for this purchase.

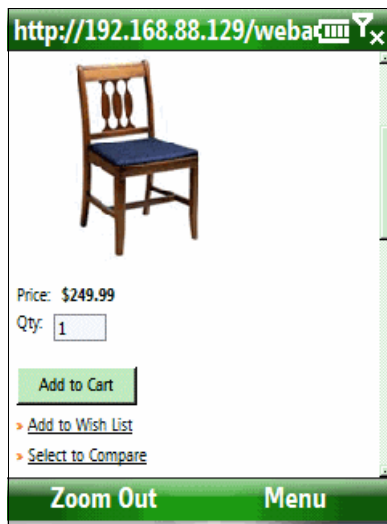


Figure 5-16 Online shopping, checkout process on Madisons Mobile Starter Store

Because the purchase meets the rules of the promotion, the discount is applied as shown in Figure 5-17.

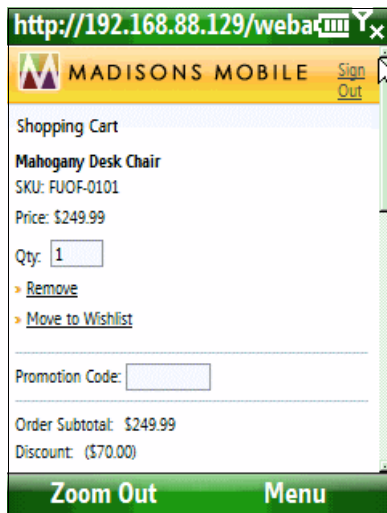


Figure 5-17 Sample promotion, discount applied

5.3 Integration with a map service provider for the mobile Store Locator

You can customize the Mobile Store Locator to integrate it with various map service providers, providing enhanced search and display capabilities on the Store Locator page. Integrating with map service providers on the Madisons mobile starter store gives an added advantage of Global Positioning System (GPS) functionality to find store locations based on the current geographic location.

Note: A strong understanding of the map service provider's API and mobile API is important to integrate its enhancements into the Store Locator successfully. Also, ensure that you complete the map service provider's sign-up process to obtain a key, application ID, or other service-specific identifier.

To integrate maps with the Store Locator in the Madisons mobile starter store:

1. Enable GPS Support:
 - a. Implement the `GPSSupport.jspf` into the JSP file in which you want to implement GPS support.
 - b. Detect whether GPS support is enabled for the current device.
 - c. Display a button that uses the current location to perform a store location search.
 - d. Before invoking the `mStoreLocatorResultView`:
 - i. Set the `geoCodeLatitude` parameter to the current location's latitude value.
 - ii. Set the `geoCodeLongitude` parameter to the current location's longitude value.
2. Integrate with the Geocoder API:
 - a. Implement the `Searchzipcode.jspf` file into the JSP file in which you want to implement the Geocoder API.
 - b. Obtain the user input from the `zipOrCity` JavaServer Pages Standard Tag Library (JSTL) variable and invoke the `geoCode` API using this input.
 - c. Parse or obtain the latitude and longitude values returned by the API provider.

- d. Set the `geoCodeLatitude` JSTL variable to contain the resulting latitude value.
 - e. Set the `geoCodeLongitude` JSTL variable to contain the resulting longitude value.
3. Integrate with a static map:
- a. Implement the `ShowMap.jspf` file into the JSP file in which you want to implement the static map.
 - b. Locate the `<div>` tag that includes the `store_map_image` value.

```
<div id="store_map_image">
<img src="" width="238" height="184"
alt="${physicalStores[i].physicalStoreIdentifier.externalIdentifier} Map" />
</div>
```
 - c. Set the value of the `image src` attribute to contain the static map URL. In the preceding code sample, the `image src` attribute contains a null value (" ") by default.
4. Integrate with specific mobile devices.

You can choose to integrate the maps with any mobile devices that support GPS and map applications.

In this book, we provide detailed steps to integrate Google Map for iPhone in the next section. For integration with any other device, for example Blackberry, refer to the manufacturer's documentation to add device-specific support.


5.3.1 Example: Google Map integration for iPhone

This example uses Google Maps API to present the integration of the mobile Store Locator with map service provider on iPhone.

Note: To use the Google Map API, you need to sign up for an API key and obtain the Map API key. A single Maps API key is valid for a single domain. You must have a Google account to get a Map API key, and the API key is connected to your Google Account. To sign up for the API, visit:

<http://code.google.com/apis/maps/signup.html>

The Madisons Mobile Starter Store does not have any integration with map service providers. The Store Locator page takes the zip code or city as the input and displays the list of stores in the entered zip code or city, as illustrated in Figure 5-18.


MADISONS MOBILE
[Sign In](#)

[Home](#) | [Store Locator](#) | Store Locator Results

Store Locator Results

Search Results: 1-3 of 3 stores found for **"Markham"**-Markham

> **Markham Centre**
 8200 Warden Ave
 Markham, Ontario
 Mon-Fri: 10am - 9pm
 Sat: 9am - 7pm
 Sun: 11am - 6pm
 > [905.413.5555](#)
 > [View map](#)
 > [Add to Store List](#)

> **Markville Mall**
 8000 McCowan Rd
 Markham, Ontario
 Mon-Fri: 10am - 9pm
 Sat: 9am - 7pm
 Sun: 11am - 6pm
 > [905.416.6666](#)
 > [View map](#)
 > [Add to Store List](#)

> **Warden Plaza**
 8250 Warden Ave
 Markham, Ontario
 Mon-Fri: 10am - 9pm
 Sat: 9am - 7pm
 Sun: 11am - 6pm
 > [905.326.8647](#)
 > [View map](#)
 > [Add to Store List](#)

Page 1/1

To manage your saved stores, use the "Store List".
 > [Store List](#)

Continue Shopping

Figure 5-18 Store Locator Results

Note: WebSphere Commerce database tables contain store location information used by the Store Locator. This information can be loaded in database using the loading utilities. You can find more information in the *Madisons Mobile Starter Store: Loading store location data into WebSphere Commerce database tables* topic in the WebSphere Commerce Version 7 Information Center.

After you load the information into database, you can modify it from WebSphere Commerce Accelerator. You can find more information in the *WebSphere Commerce Accelerator: Changing store location information* topic in the WebSphere Commerce Version 7 Information Center:

<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.user.doc/tasks/tstlocin.htm>

To implement map integration, you need to:

1. Enable GPS support.
2. Integrate with Google static map.
3. Integrating with the iPhone map application.

Enable GPS support

This section describes the steps to enable the GPS support in the Madisons Mobile Starter Store for iPhone. You implement the `GPSSupport.jspf` file in the `stores/WebContent/Madisons/mobile/snippets/StoreLocator` directory as explained in following steps:

1. Create the `GPSSupport.jspf` skeleton file as shown in Example 5-2.

Example 5-2 GPSSupport.jspf skeleton

```
<script type="text/javascript">
//
    //Add a check if geolocation API is supported
    //Add a function to capture the current location
    //Add a function to set the form elements with the captured
    location and submit the form
    //display the form
//]]&gt;
&lt;/script&gt;</pre><hr/></div><div data-bbox="351 942 850 961" data-label="Page-Footer"><p>Chapter 5. Mobile commerce features in WebSphere Commerce V7 <b>211</b></p></div>
```

2. Verify that the geolocation API is supported in the handset that is used. Replace the statement “Add a check if geolocation API is supported” shown in Example 5-2 with the code snippet shown in Example 5-3.

Example 5-3 Check for geolocation API support

```
if (navigator.geolocation) {  
    //Add a function to capture the current location  
    //Add a function to set the form elements with the captured  
    location and submit the form  
    //display the form  
}
```

3. Capture the current location of the handset. You can set the maximum age, which is used if the GPS cannot locate the current position. Replace the statement “Add a function to capture the current location” shown in Example 5-3 with the code snippet shown in Example 5-4.

Example 5-4 Capture current location

```
function captureCurrentLocation() {  
    // Get location no more than 10 minutes old. 600000 ms = 10  
minutes.  
    navigator.geolocation.getCurrentPosition(showLocation, showError,  
    {enableHighAccuracy:true,maximumAge:600000});  
}  
function showError(error) {  
    //If there is any error in capturing the current location, it  
    will throw an alert to the user.  
    alert(error.code + ' ' + error.message);  
}
```

4. Use the location captured in step 3 to populate the form with the latitude and longitude of the current location. After capturing the location, submit the form to get the search results based on the query that was entered. Replace the statement “Add a function to set the form elements with the captured location and submit the form” in shown in Example 5-3 with the lines of code shown in Example 5-5.

Example 5-5 Code snippet to populate the GPS form

```
function showLocation(position) {  
    var fromForm = document.getElementById("store_locator_gps_form");  
    var geoCodeLatitude = fromForm.geoCodeLatitude;  
    var geoCodeLongitude = fromForm.geoCodeLongitude;  
  
    geoCodeLatitude.value = position.coords.latitude;
```

```

        geoCodeLongitude.value = position.coords.longitude;

        fromForm.action = "mStoreLocatorResultView";
        fromForm.submit();
    }

```

Example 5-6 gives sample code for the form that is used in the Madisons Mobile Starter Store. It takes the search radius as input from the user. Replace the line “display the form” in Example 5-3 on page 212 with the lines of code in Example 5-6.

Example 5-6 Code snippet to implement the form

```

this.document.write("<form id='store_locator_gps_form' method='post'
action='>");

this.document.write("<input type='hidden' id='storeId' name='storeId'
value='${WCParam.storeId}' />");

this.document.write("<input type='hidden' id='langId' name='langId'
value='${WCParam.langId}' />");

this.document.write("<input type='hidden' id='catalogId'
name='catalogId' value='${WCParam.catalogId}' />");

this.document.write("<input type='hidden' id='productId'
name='productId' value='${WCParam.productId}' />");

this.document.write("<input type='hidden' id='fromPage' name='fromPage'
value='${WCParam.fromPage}' />");

this.document.write("<input type='hidden' id='geoCodeLatitude'
name='geoCodeLatitude' value='' />");

this.document.write("<input type='hidden' id='geoCodeLongitude'
name='geoCodeLongitude' value='' />");

this.document.write("<input type='hidden' id='uom' name='uom'
value='KTM' />");

this.document.write("<input type='hidden' id='whichSearch'
name='whichSearch' value='useGPS' />");

this.document.write("<div><fmt:message key='MSTLOC_FIND_ME_MESSAGE'
bundle='${storeText}' /></div>");

```

```

this.document.write("<label for='radius'><div><fmt:message
key='MSTLOC_SELECT_RADIUS' bundle='${storeText}' /></div></label>");

this.document.write("<div><select id='radius' name='radius'
class='coloured_input'>");

this.document.write("<option value='5'><fmt:message
key='MSTLOC_RADIUS_5' bundle='${storeText}' /></option>");

this.document.write("<option value='10'><fmt:message
key='MSTLOC_RADIUS_10' bundle='${storeText}' /></option>");

this.document.write("<option value='15'><fmt:message
key='MSTLOC_RADIUS_15' bundle='${storeText}' /></option>");

this.document.write("<option value='20'><fmt:message
key='MSTLOC_RADIUS_20' bundle='${storeText}' /></option>");

this.document.write("      </select> <fmt:message
key='MSTLOC_RADIUS_UNIT' bundle='${storeText}' /></div>");

this.document.write("<input type='submit' id='find_me' name='find_me'
onclick='javascript:captureCurrentLocation(); return false;'
class='input_button' value='<fmt:message key='MSTLOC_FIND_ME'
bundle='${storeText}' />' />");

this.document.write("<div class='bold'><fmt:message key='MSTLOC_OR'
bundle='${storeText}' /></div>");

this.document.write("</form>");

```

On an iPhone, the Store Locator page looks similar to that shown in Figure 5-19 after implementing the GPSSupport.jspf file.

Figure 5-19 Store locator

Note: For more information about the geolocation API for iPhone, refer to:
<http://developer.apple.com/iPhone/library/navigation/index.html>

Integrate with Google static map

Implement the ShowMap.jspf file located in the stores/WebContent/Madisons/mobile/snippets/StoreLocator directory as shown in Example 5-7. Include this code in the JSP where you want to display the map for a single store or multiple stores.

Example 5-7 Display Google map for a single store

```
<c:set var="latLong"
value="\${physicalStores[i].locationInfo.geoCode.latitude},\${physicalStores[i].locationInfo.geoCode.longitude}" />

<c:url var="staticMapURL" value="http://maps.google.com/staticmap">
  <c:param name="center" value="\${latLong}" />
  <c:param name="zoom" value="12" />
  <c:param name="size" value="238x184" />
  <c:param name="maptype" value="mobile" />
  <c:param name="markers" value="\${latLong},redc" />
  <c:param name="key" value="{MAP_KEY}" />
</c:url>

<div id="store_map_image"></div>
```

To complete the process of implementing the map that you want to display for a single store or multiple stores, follow these steps:

1. Obtain the latitude and longitude values for the store that you want to locate on map.
2. Create the URL to Google static map. You can find more information about the URL the Google Maps API.
 - a. Pass the comma separated value of latitude and longitude to the “center” parameter.
 - b. Set the zoom to 12. You can set the zoom to any value between 1 and 19, based on your requirement.
 - c. Set the size to 238x184. You can set the size to any value, depending on your requirement.
 - d. Set maptype as mobile.
 - e. Set markers to the latitude, longitude, and marker color plus alphanumeric indicator of all the locations that you want to mark on the map. If you want to plot multiple locations on the map, then separate these values with a vertical bar (|) character, as shown in Example 5-8.
 - f. Set key to the value that you received after you registered your domain with Google.
3. Locate the <div> tag that includes the store_map_image value. Set the value of attribute to contain the static map URL. See Example 5-8.

Example 5-8 Implement static map URL

```
<c:set var="markers" value="" />  
<c:set var="latLong" value="${geoCodeLatitude},${geoCodeLongitude}" />  
<c:forEach var="i" begin="0" end="${resultStoreNum-1}">  
  <c:choose>  
    <c:when test="${markers == ''}">  
      <c:set var="markers"  
value="${physicalStores[i].locationInfo.geoCode.latitude},${physical  
Stores[i].locationInfo.geoCode.longitude},green${i+1}" />  
    </c:when>  
    <c:otherwise>  
      <c:set var="markers"  
value="${markers}|${physicalStores[i].locationInfo.geoCode.latitude}  
,$${physicalStores[i].locationInfo.geoCode.longitude},green${i+1}" />
```

```
        </c:otherwise>
    </c:choose>
</c:forEach>
<c:url var="staticMapURL" value="http://maps.google.com/staticmap">
    <c:param name="center" value="{latLong}" />
    <c:param name="zoom" value="9" />
    <c:param name="size" value="238x184" />
    <c:param name="maptype" value="mobile" />
    <c:param name="markers" value="{markers}" />
    <c:param name="key" value="{MAP_KEY}" />
</c:url>

<div id="store_map_image"></div>
```

As a result of integration with Google static map, the search results page for the Madisons Mobile Starter Store looks similar to that shown in Figure 5-20.


MADISON'S MOBILE
[Sign In](#)

[Home](#) | [Store Locator](#) | Store Locator Results

Store Locator Results

Search Results: 1-3 of 3 stores found for "markham"-Markham



- Markham Centre**
 8200 Warden Ave
 Markham, Ontario
 Mon-Fri: 10am - 9pm
 Sat: 9am - 7pm
 Sun: 11am - 6pm
[905.413.5555](#)
[View map](#)
[Add to Store List](#)
- Markville Mall**
 8000 McCowan Rd
 Markham, Ontario
 Mon-Fri: 10am - 9pm
 Sat: 9am - 7pm
 Sun: 11am - 6pm
[905.416.6666](#)
[View map](#)
[Add to Store List](#)
- Warden Plaza**
 8250 Warden Ave
 Markham, Ontario
 Mon-Fri: 10am - 9pm
 Sat: 9am - 7pm
 Sun: 11am - 6pm
[905.326.8647](#)
[View map](#)
[Add to Store List](#)

Page 1/1

[Continue Shopping](#)

Figure 5-20 Store Locator results

Integrating with the iPhone map application

You can integrate the mobile storefront with the iPhone map application. On the Store Locator results page, when a user clicks **View Map**, control is taken to the iPhone map application. You can implement this functionality as follows:

1. Update the ViewMap.jspf file to contain the required logic to invoke the default iPhone map application with the code snippet shown in Example 5-9.

Example 5-9 Code snippet for ViewMap.jspf file

```
<c:choose>
  <c:when test="${!empty physicalStore}">
    <c:set var="thePhysicalStore" value="{physicalStore}" />
  </c:when>
  <c:otherwise>
    <c:set var="thePhysicalStore" value="{physicalStores[i]}" />
  </c:otherwise>
</c:choose>

<c:if test="${fn:startsWith(header['User-Agent'], 'iPhone')}">
  <c:set var="storeLatitude"
value="{thePhysicalStore.locationInfo.geoCode.latitude}" />
  <c:set var="storeLongitude"
value="{thePhysicalStore.locationInfo.geoCode.longitude}" />
  <c:set var="storeAddress"
value="{thePhysicalStore.locationInfo.address.addressLine[0]},
{thePhysicalStore.locationInfo.address.city},
{thePhysicalStore.locationInfo.address.stateOrProvinceName}" />
  <c:set var="storeName"
value="({thePhysicalStore.physicalStoreIdentifier.externalIdentifi
fier})" />

  <c:url var="mapsGoogleURL" value="http://maps.google.com/maps">
    <c:param name="v" value="2" />
    <c:param name="t" value="mobile" />
    <c:param name="mrt" value="loc" />
    <c:param name="ll" value="{storeLatitude},{storeLongitude}"
/>
    <c:param name="q" value="{storeAddress} {storeName}" />
  </c:url>
  <li><span class="bullet">&#187; </span><a
href="{fn:escapeXml(mapsGoogleURL)}"><fmt:message
key='MST_VIEW_MAP' bundle='{storeText}' /></a></li>
</c:if>
```

2. Detect the device as an iPhone with the following line:

```
<c:if test="${fn:startsWith(header['User-Agent'], 'iPhone')}"/>
```
3. Obtain the store location information from the store location variable with the following line:

```
physicalStores[i].locationInfo.address.addressLine[0] returns  
address line 1 of the store location.
```
4. Construct the Google Maps URL.
5. Include the ViewMap.jspf file in the StoreLocatorResults.jsp file using the following line:

```
<%@ include file="../../Snippets/StoreLocator/ViewMap.jspf" %>
```

After completing these steps, when a user clicks **View Map** in the Search results page, the map is displayed as shown in Figure 5-21.

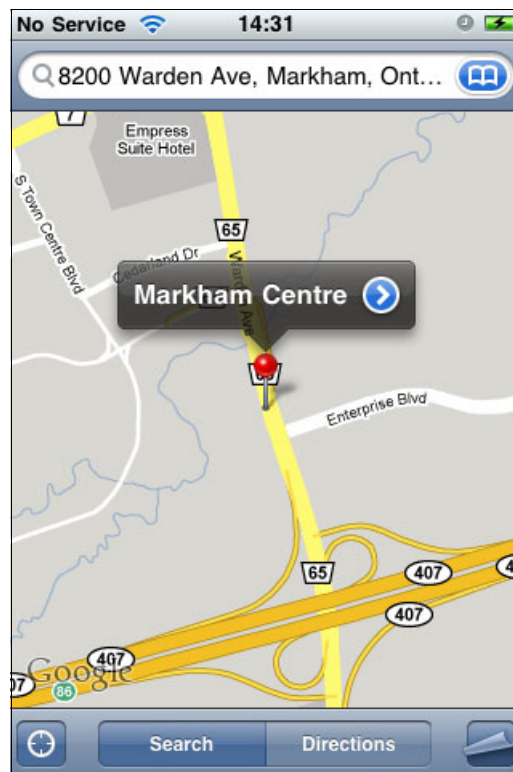


Figure 5-21 Map results

5.4 Buy on mobile device and select shipping address

As mentioned in “Functions not included in the Madisons Mobile Starter Store” on page 51, the version of Madisons Mobile Starter Store that ships with IBM WebSphere Commerce V7 includes only checkout pages that allow shoppers to buy online and pick up in a store. As such, a shopper cannot buy on the mobile device and ship the products to a selected address. This type of functionality is also known as *endless aisle in the store*, because it allows the shopper to browse and check out products that might be available online but not in the store, while the shopper is in a brick-and-mortar store.

In this section, we explain how you can amend the Madisons Mobile Starter Store with endless aisle functionality.

Note: The steps in this section assume that you have the Madisons starter store published with the Madisons mobile add-on starter store. The Madisons mobile starter store is published by default on a standard installation of IBM WebSphere Commerce V7 Developer Edition. However, if you have cleaned your toolkit using the `resetstores` and `resetdb` scripts, you need to publish the `Madisons.sar` followed by the `MadisonsMobile.sar` to complete the steps in the following sections.

5.4.1 Standard checkout page flow for Madisons Mobile Starter Store

The standard page flow for checkout by guest shoppers that have not yet selected a preferred store in Madisons Mobile Starter Store, as it ships with IBM WebSphere Commerce V7, is outlined in Figure 5-22 on page 223. The numbers in Figure 5-22 refer to the following summary:

1. The product page where the shopper clicks **Add to Cart** to add the product to the Shopping Cart.
2. The Shopping Cart that is presented after a shopper clicks **Add to Cart** or selects the **Shopping Cart** link that is present in the footer of all pages. The shopper can click **Proceed to Checkout** to transition to the next page.

There are two versions of this page:

- The regular Shopping Cart that shows the products that the shopper has added.
- The page for an empty cart, showing a message that the shopper has not added products to the cart.

Note: The Shopping Cart page for the standard Madisons mobile starter store in IBM WebSphere Commerce V7 has a single radio button, Pick Up at Store, just above the Proceed to Checkout button. We will add an additional radio button that allows shoppers to elect to have the ordered items shipped to a shipping address of their choice.

3. The Sign in or Checkout page allows registered shoppers to sign in to retrieve settings or guest shoppers to check out without signing in. In this scenario, the shopper clicks **Continue Checkout** to check out as a guest shopper.
4. The Store Locator is where the shopper enters a zip or postal code or a city name and clicks **Continue Checkout** to transition to the next page.
5. The Store Locator search results lists store that match the search terms entered on the Store Locator page. Clicking **Add to Store List** and then **Continue Checkout** opens the next page.
6. The Your Store List opens again with the stores selected in the previous page. On this page, shoppers select the store from which to pick up products, and then click **Continue Checkout**.
7. The billing address form opens. Shoppers can enter billing information, and click **Continue Checkout**.
8. After editing the billing address, the billing address selection page is shown with the entered address. The shopper can then select the address and click **Continue Checkout**.

Note: In this scenario, we check out with a new guest shopper account. If using an existing account with a predefined addresses, the billing address selection page is shown at step 7. Whether this page is shown is controlled using the conditional JavaScript on the billing address selection page.

9. The payment method page opens, from which shoppers can select the preferred payment method, including the Pay in Store method, enter the payment details, and click **Continue Checkout**.
10. The order summary page opens. This page is the final page before the order is submitted. This page allows the shopper to verify the pick-up location, billing address, payment method, and order details before finalizing the order by clicking **Place Your Order**.
11. After the order is submitted, the order confirmation page shows the final confirmation that the order is received and is being processed.

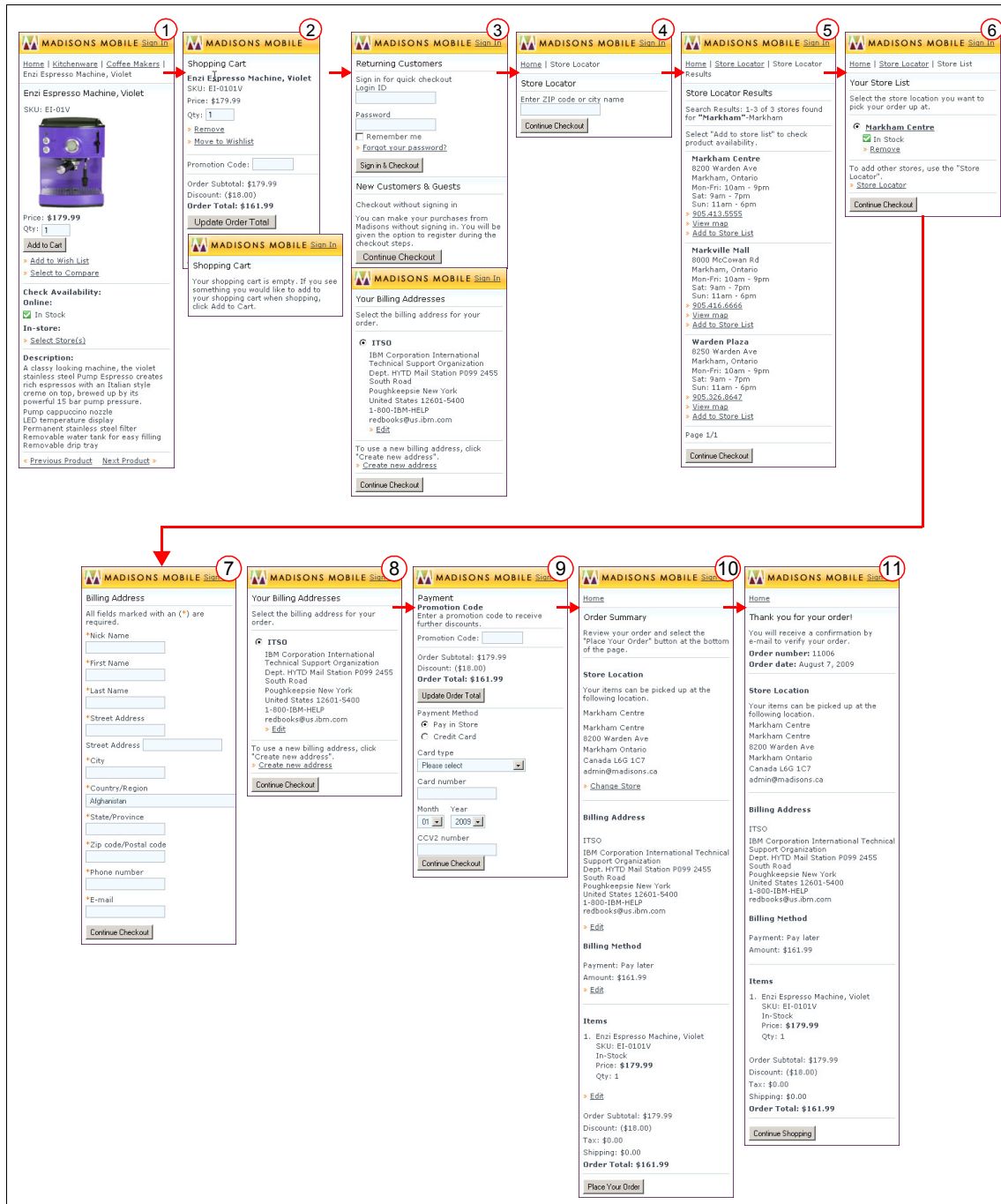


Figure 5-22 Standard page flow in Madisons Mobile Starter Store

5.4.2 Checkout with shipping for Madisons Mobile Starter Store

As mentioned earlier, we will amend the process outlined in 5.4.1, “Standard checkout page flow for Madisons Mobile Starter Store” on page 221, to provide the shopper the option to ship products to a selected address. This alternate scenario is outlined in Figure 5-23. The main difference between the flow in Figure 5-22 on page 223 and that in Figure 5-23 is that the store selection pages are replaced with shipping selection pages.

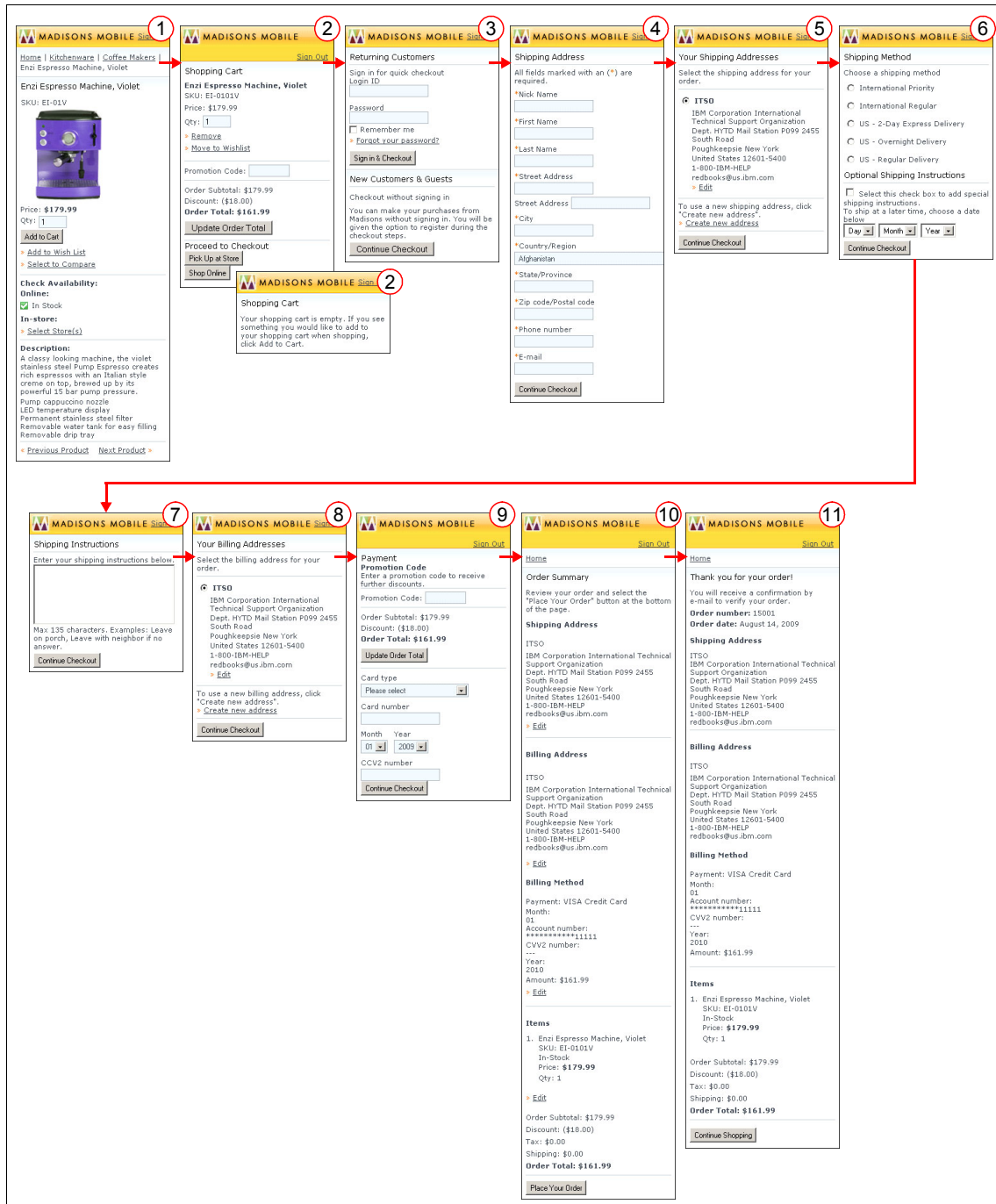


Figure 5-23 Page flow for the mobile checkout with shipping

The numbers in Figure 5-23 refer to the following summary:

1. The product page is the page from where the shopper can click **Add to Cart** to add the product to the shopping cart. We will not change this page.
2. The shopping cart is the page that is presented after a shopper clicks **Add to Cart** or selects the **Shopping Cart** link that is present in the footer of all pages. The shopper can click **Proceed to Checkout** to transition to the next page.

There are two versions of this page:

- The regular shopping cart that shows the products that the shopper has added. We will change this page to add the option of selecting a shipping option in addition to the standard BOPIS option. Figure 5-24 shows a comparison of the standard page and the new page.
- This is a page for an empty cart that shows a message that the shopper has yet to add products to the cart. We will not change this page.

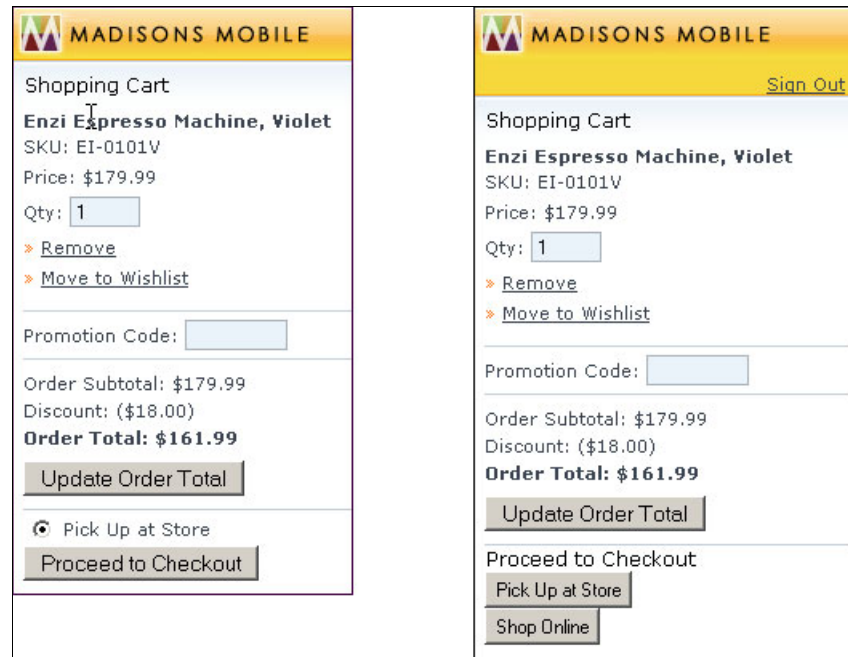


Figure 5-24 The standard (left) and the new shopping cart page (right)

3. The next page is the Sign in or Checkout page, which allows registered shoppers to sign in to retrieve settings or guest shoppers to check out without signing in. In this scenario, the shopper clicks **Continue Checkout** to check out as a guest shopper. We will not change this page.

4. The new shipping address page opens from where the shopper can enter shipping information and click **Continue Checkout**.
5. After editing the shipping address, the shipping address selection page opens and shows the entered address. The shopper can then select the address and click **Continue Checkout**.

Note: In this scenario, we check out with a new guest shopper account. If using an existing account with a predefined addresses, the shipping address selection page is shown at step 4. Whether this page is shown is controlled using conditional JavaScript on the shipping address selection page.

6. The new shipping options page opens from where the shopper can select the courier, delivery speed, an optional shipping date, and whether there are special shipping instructions.
7. The new shipping instructions page is shown only if the shopper selects to specify shipping instructions on the previous page. Here the shopper can specify free-text instructions, such as “Leave on porch if nobody is home.”
8. The standard billing address selection page allows the shopper to select the billing address. In our scenario, the shopper has already created a shipping address, so the shopper can select this as the billing address as well and click **Continue Checkout**.
9. The payment method page allows the shopper to select the preferred payment method and to enter the payment details, and then click **Continue Checkout**. Note that because we are not picking up in the store, the pay in store method is not on this page.
10. This is the order summary page, which is the final page before the order is submitted. This page allows the shopper to verify the pick-up location, billing address, payment method, and order details before finalizing the order by clicking **Place Your Order**. We will change this page to display the shipping address rather than the store address if selected.
11. After the order is submitted, the order confirmation page shows the final confirmation that the order is received and is being processed. We will change this page to display the shipping address rather than the store address if selected.

5.4.3 Analyze the existing Madisons Mobile Starter Store code

As described in 5.4.2, “Checkout with shipping for Madisons Mobile Starter Store” on page 224, we need to modify a number of pages as well as introduce some new pages. In this section, we analyze the existing pages to determine the files that we need to modify.

Summary of modified pages for shipping checkout

In the overview of the current pick up in store checkout flow and the online shopping flow, we noted that we need to modify the pages as follows:

- ▶ Shopping cart (not empty)
Add an option to Shop Online in addition to the existing Pick Up at Store option.
- ▶ Payment method
Ensure that Pay in Store is removed when choosing the Shop Online option on the shopping cart page. It is apparent in the analysis of this page that we also need to modify it to support separate shipping and billing addresses.
- ▶ Order summary
Add logic to display the shipping address and options when appropriate.
- ▶ Order confirmation
Add logic to display the shipping address and options when appropriate.
- ▶ Breadcrumb trail include
The Madisons mobile starter store generates the breadcrumb trail on top of all pages from a central include file names BreadCrumbTrailDisplay.jspf. We need to amend this file to ensure that the breadcrumb trail on new pages, as well as on existing pages in the modified flow, is shown correctly.
- ▶ Sign in or checkout page
We need to update this page to redirect correctly in the new checkout flow.

Summary of new pages for shipping checkout

In addition to the modifications that we have outlined thus far, we need to create the following new pages:

- ▶ Shipping address selection
Similar to the billing address selection page.
- ▶ Shipping address create/edit
Similar to the billing address create/edit page.

- ▶ Shipping method selection
A new page for the shopper to select the shipping options.
- ▶ Shipping instructions create/edit
A new page for the shopper to enter their delivery instructions.

In the following sections, we analyze the existing pages to understand how we can best implement these changes.

Analyze the current checkout pages

In this section, we go through the existing checkout process for Madisons Mobile Starter Store to analyze the existing pages to determine the best approach to add shipping address handling to the process:

1. Open the development environment by selecting **Start** ∅ **Programs** ∅ **IBM** ∅ **WebSphere** ∅ **WebSphere Commerce** ∅ **WebSphere Commerce Developer Enterprise**.
2. Switch to the Java EE perspective if it is not already active:
 - a. Select **Window** ∅ **Open Perspective** ∅ **Other**.
 - b. The Open Perspective window opens. Select **Java EE**, and click **OK**.

3. In the Servers view, right-click **WebSphere Commerce Test Server**, and select **Start**, as shown in Figure 5-25.

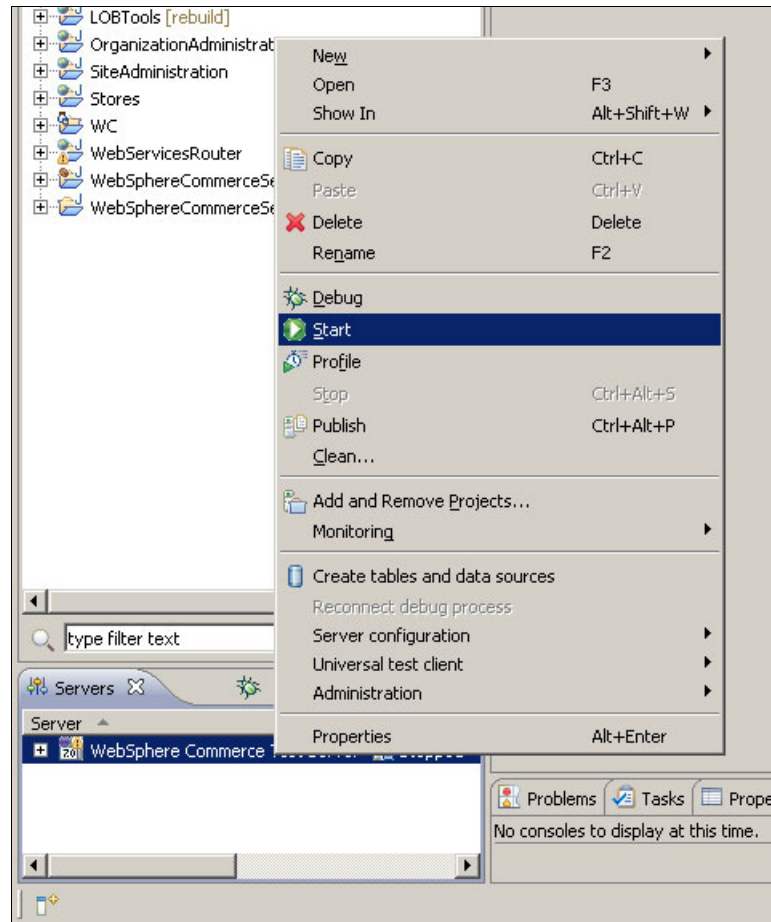


Figure 5-25 Starting the server

The server state changes as shown in Figure 5-26.



Figure 5-26 Server in the starting state

After a while, the server status changes to *Started*, as shown in Figure 5-27. In our case, the server took approximately 6 minutes to start.

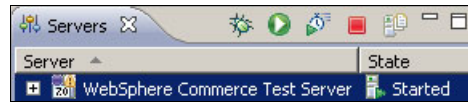


Figure 5-27 Server in the started state

4. Open a Web browser and navigate to a product page. We used Mozilla Firefox to navigate to a product in the catalog.

Important: Although you are browsing mobile pages, you need to use a desktop Web browser in this example, because you must inspect details about the application, such as the source code, which are not usually available on a mobile browser or simulator.

Also, ensure that you perform these steps in a new Web browser instance, because the flow assumes that you are checking out as a guest user with no existing addresses in the address book.

We chose the violet Enzi Espresso Machine at the following URL:

`http://localhost/webapp/wcs/stores/servlet/mProduct1_10051_10051_-1_10074_10345_10053_10053_catNav__`

Note: This link worked in our environment with the standard starter stores deployed at the time of installing IBM WebSphere Commerce V7 Developer Edition. If you published the Madisons Starter Store manually or changed the catalog, this link might not work for you. In this case, navigate the catalog to find a product that is in stock both online and in one of the brick-and-mortar stores. The violet Enzi Espresso Machine is in the Coffee Makers subcategory of the Kitchenware top category.

5. From the product page click **Add to Cart**.
6. The shopping cart page opens with the product shown in the cart. Because we need to add an extra radio button for shipping checkout, we must determine the JSP that renders this page. Thus, we note the URL that shows in the browser's address bar.

In our scenario, we saw the following URL (everything following the `krypto` parameter in bold formatting is excluded for brevity):

`https://localhost/webapp/wcs/stores/servlet/mOrderItemDisplay?productId=12245&catalogId=10301&categoryId=10174&langId=-1&storeId=10051`

We highlighted the relevant information that we use later to determine to which JSP this view name corresponds.

Note: The exact values and order of parameters might be different in your configuration.

7. Now, we need to determine how we can add another radio button by analyzing the source code of the page as follows:

a. Use your browser's method for viewing the source code:

- In Mozilla Firefox, select **View** \varnothing **Page Source**.
- In Microsoft® Internet Explorer, select **View** \varnothing **Source**.

We used Mozilla Firefox, so we selected **View** \varnothing **Page Source**.

b. The page source opens in a separate window. Search for the text on the existing radio button, for example:

Pick Up at Store

In the standard page source window in Mozilla Firefox, you can search by pressing Ctrl+F.

The code snippet shown in Example 5-10 should display. (We reformatted the code in this example to improve readability.)

Example 5-10 The code for the submit button on the Shopping Cart page

```
<div id="shipping_options">
  <div class="radio_container">
    <input type="radio" name="shipping_option"
      id="pick_up_at_store" checked />
    <label for="pick_up_at_store">Pick Up at Store</label>
  </div>
  <button type="button"
onclick="window.location.href='http://localhost/webapp/wcs/stores/se
rvlet/mCheckoutLogon?catalogId=10051&langId=-1&storeId=10051'">Proce
ed to Checkout</button>
</div>
```

The highlighted code in Example 5-10 shows that the Madisons mobile starter store is designed to always go to the sign in or check out page. Thus, that page must control the next page in the checkout flow.

To add the additional Shop Online option, we need to be able to control to which page the mCheckoutLogon page redirects. We discuss the possible options for doing this task later in 5.4.4, “Design new shopping flow” on page 239.

8. Close the source view, and click **Proceed to Checkout** to continue the checkout process.
9. The sign in or checkout page opens. Click **Continue Checkout** to check out as a guest shopper.
10. The Store Locator opens. Enter Markham in the entry field, and click **Continue Checkout**.
11. A list of stores in Markham opens. Find the Markham Centre entry, and click **Add to Store List**.
12. The page refreshes, and the link under Markham Centre changes to read "Remove from Store List" to signify that the store was added. Scroll down to the bottom of the page, and click **Continue Checkout**.
13. You are presented with the store list with the Markham Centre location as the only option. Select **Markham Centre**, and click **Continue Checkout**.
14. The billing address details page opens. Because you need to create a page very similar to this page, take note of the URL of this page from the browser's address bar. In our scenario, we noted the following URL:

```
http://localhost/webapp/wcs/stores/servlet/mOrderBillingDetails?catalogId=10051&orderId=11007&langId=-1&storeId=10051
```

15. In the billing address page, enter a new billing address, and click **Continue Checkout**. We entered the following information:

Nick Name:	ITS0
First Name:	IBM Corporation
Last Name:	International Technical Support Organization
Street Address 1:	Dept. HYTD Mail Station P099
Street Address 2:	2455 South Road
City:	Poughkeepsie
Country/Region:	United States
State/Province:	New York
Zip code/ Postal code:	12601-5400
Phone number:	1-800-IBM-HELP
E-mail:	redbooks@us.ibm.com

16. The billing address selection page opens with the address that you just entered. Because you need to create a page very similar to this page, take note of the URL from the browser's address bar. We noted the following URL. We left out the `krypto` parameter for brevity:

```
https://localhost/webapp/wcs/stores/servlet/mOrderBillingAddressSelection?catalogId=10051&langId=-1&storeId=10051
```

Because the page does not have `addressId` parameters, the JSP must retrieve the shopper's address book information in the JSP directly.

17. Select the billing address that was created in step 14, and click **Continue Checkout**. We selected **ITSO**, and clicked **Continue Checkout**.
18. The payment method page opens. Take note of the URL of this page from the browser's address bar. In our example, we saw the following URL:


```
https://localhost/webapp/wcs/stores/servlet/m0OrderPaymentDetails?langId=-1&storeId=10051&catalogId=10051&addressId=11402
```

We will need this URL at a later step in the analysis where we analyze the struts configuration.

Note that the address ID is passed from the billing address selection page to this page. This address ID shows that the billing address is most likely not updated by the billing address selection page itself but, rather, is updated as part of a later step. This information is important, because we need to determine how best to add the shipping address to the order.

In your environment, the addressId parameter value will likely be different.

19. Because we observed that the billing address information has not yet been submitted to a controller command or an order service, we need to verify whether this page performs the update by viewing the page source of the current page using the browser's view source function.

We used Mozilla Firefox, so we selected **View**  **Page Source**.

20. A page source window opens. Search for the following text:

Continue Checkout

We pressed Ctrl+F and then entered Continue Checkout in the search bar at the bottom. The HTML code in Example 5-11 is displayed.

Example 5-11 Continue Checkout button code from the payment details page

```
<input type="button" name="proceed_to_checkout"
id="proceed_to_checkout" onclick="submitPaymentInfo();"
value="Continue Checkout" />
```

The highlighted code in Example 5-11 shows that the JavaScript function submitPaymentInfo is invoked when the shopper clicks the button. We next need to analyze the source code of this function.

21. Search for the text function submitPaymentInfo. We pressed Ctrl+F and then entered function submitPaymentInfo in the search bar at the bottom. The HTML code in Example 5-12 displayed.

Example 5-12 The source code for the submitPaymentInfo JavaScript function

```
function submitPaymentInfo() {
    var paymentMethods = document.getElementsByName('payMethodId_radio');
    for(var i = 0; i < paymentMethods.length; i++) {
```



```

    if(paymentMethods[i].checked) {
        if(paymentMethods[i].value == "credit_card") {
            document.getElementById("payMethodId_creditcard").value =
                document.getElementById("card_type").value;
            document.getElementById("payment_method_form_creditcard").submit();
        }
        else {
            document.getElementById("payMethodId_payinstore").value =
                paymentMethods[i].value;
            document.getElementById("payment_method_form_payinstore").submit();
        }
    }
}
}
}

```

The highlighted code in Example 5-12 shows that, depending on the payment type, either the `payment_method_form_creditcard` or the `payment_method_form_payinstore` form is submitted. We analyze the latter to understand the target for the form and the parameters submitted.

22. Search for the text `id="payment_method_form_payinstore"`. We pressed Ctrl+F and then entered `id="payment_method_form_payinstore"` in the search bar at the bottom. The HTML code in Example 5-13 displayed.

Example 5-13 HTML source code for the `payment_method_form_payinstore` form

```

<form id="payment_method_form_payinstore" action="OrderChangeServicePIAdd">
    <fieldset>
        <input type="hidden" name="langId" value="-1" />
        <input type="hidden" name="storeId" value="10051" />
        <input type="hidden" name="catalogId" value="10051" />
        <input type="hidden" name="payMethodId" value="" id="payMethodId_payinstore" />
        <input type="hidden" name="URL"
value="http://localhost/webapp/wcs/stores/servlet/mOrderShippingBillingSummaryView?ca
talogId=10051&langId=-1&storeId=10051"/>
        <input type="hidden" name="piAmount" value="161.99000"/>
        <input type="hidden" name="billing_address_id" value="11902"/>
        <input type="hidden" name="addressId" value="11902"/>
        <input type="hidden" name="errorViewName" value="mOrderPaymentDetails"/>
        <input type="hidden" name="authToken"
value="3%2czzgqxWRhBRPd%2b4HtQmI%2fwb3TtG4%3d" />
    </fieldset>
</form>

```

The HTML in Example 5-13 shows that the information in the hidden fields within the form is submitted to the `OrderChangeServicePIAdd` struts action.

This action adds a payment instruction to the order with the specified payment information. We notice that both the addressId and the billing_address_id are specified as parameters. However, the action uses only the billing_address_id.

Also, we note that the authToken request parameter is specified as a hidden field in the form as part of the cross-site request forgery (CSRF) protection feature, which works only for SSL protected pages. As such, we must ensure that the pages that we introduce require SSL.

23. Close the source code window. You can use Ctrl+W to close the window.

24. On the payment method page, select **Pay in Store**, and click **Continue Checkout**.

25. The order summary page opens as shown in Figure 5-28. Make a note of the URL in the browser's address bar. We saw the following URL. We left out the krypto parameter for brevity:

`http://localhost/webapp/wcs/stores/servlet/mOrderShippingBillingSummaryView?catalogId=10051&langId=-1&storeId=10051`

We need to change this JSP to display the shipping address in place of the store location if the shopper has selected a shipping address.


 MADISONS MOBILE Sign In		
Home		
Order Summary		
Review your order and select the "Place Your Order" button at the bottom of the page.		
Store Location	Billing Address	Payment: Pay later
Your items can be picked up at the following location.	ITSO	Amount: \$161.99
Markham Centre	IBM Corporation International Technical Support Organization	Edit
Markham Centre	Dept. HYTD Mail Station P099 2455	
8200 Warden Ave	South Road	Items
Markham Ontario	Poughkeepsie New York	1. Enzi Espresso Machine, Violet
Canada L6G 1C7	United States 12601-5400	SKU: EI-0101V
admin@madisons.ca	1-800-IBM-HELP	In-Stock
Change Store	redbooks@us.ibm.com	Price: \$179.99
	Edit	Qty: 1
	Billing Method	Edit
	Payment: Pay later	Order Subtotal: \$179.99
	Amount: \$161.99	Discount: (\$18.00)
	Edit	Tax: \$0.00
		Shipping: \$0.00
		Order Total: \$161.99
		Place Your Order

Figure 5-28 The order summary page (shown here split in three parts)

26. Click **Place Your Order**.

27. The order confirmation page opens. Make a note of the address of this page. In our example, we noted down the following URL. We left out the `krypto` and other parameters for brevity:

`https://localhost/webapp/wcs/stores/servlet/mOrderShippingBillingConfirmationView?catalogId=10051&langId=-1&storeId=10051`

We will change the order confirmation page in the same way as the order summary page, replacing the store location with the shipping address if one was selected during checkout.












We now have the information necessary to change the pages. Table 5-6 summarizes our analysis.

Table 5-6 Values discovered during the analysis

Information	Value
Store ID	10051
Shopping Cart view	mOrderItemDisplay
Billing address selection view	mOrderBillingAddressSelection
Billing address create view	mOrderBillingDetails
Payment details view	mOrderPaymentDetails
Order summary view	mOrderShippingBillingSummaryView
Order confirmation view	mOrderShippingBillingConfirmationView

Analyze struts configuration for checkout pages

We now use the information that we gathered in our analysis to determine the JSPs that we need to modify by analyzing the struts configuration files as follows:

1. Open the development environment by selecting **Start**  **Programs**  **IBM**  **WebSphere**  **WebSphere Commerce**  **WebSphere Commerce Developer Enterprise**.
2. Switch to the Java EE perspective if it is not already active:
 - a. Select **Window**  **Open Perspective**  **Other**.
 - b. The Open Perspective window opens. Select **Java EE**, and click **OK**.
3. In the Enterprise Explorer, expand and right-click **Stores**  **WebContent**  **WEB-INF**  **struts-config-ext.xml**.
4. From the context menu, select **Open With**  **Text Editor**.


5. The `struts-config-ext.xml` file opens in a text editor. For each of the views listed in Table 5-6, complete the following steps:
 - a. Press Ctrl+F to open the Find/Replace dialog box.
 - b. In the Find/Replace window, enter the first view name, for example, `mOrderItemDisplay`, ensure that **Wrap search** is selected, and click **Find**.

An excerpt similar to Example 5-14 is displayed. (We reformatted the XML in Example 5-14 for improved readability.)

Note that the store ID after the view name (the two are separated with a forward slash character) corresponds to the store ID that we noted in Table 5-6.
 - c. Note the JSP name (highlighted in bold text in Example 5-14).

Example 5-14 Global forward for the `mOrderItemDisplay` view

```
<forward
  className="com.ibm.commerce.struts.ECActionForward"
  name="mOrderItemDisplay/10051"
  path="/mobile/ShoppingArea/ShopcartSection/OrderItemDisplay.jsp">
</forward>
```

6. Repeat steps 1 through 5 for the remaining view names listed in Table 5-6.
7. Close the `struts-config-ext.xml` editor by selecting **File**  **Close**.

Your completed a table should be similar to that shown in Table 5-7.

Table 5-7 Mapping from view name to JSP for pages that need modification

View Name	JSP Location
<code>mOrderItemDisplay</code>	<code>/mobile/ShoppingArea/ShopcartSection/OrderItemDisplay.jsp</code>
<code>mOrderBillingAddressSelection</code>	<code>/mobile/ShoppingArea/CheckoutSection/OrderBillingAddressSelection.jsp</code>
<code>mOrderBillingDetails</code>	<code>/mobile/ShoppingArea/CheckoutSection/OrderBillingDetails.jsp</code>
<code>mOrderPaymentDetails</code>	<code>/mobile/ShoppingArea/CheckoutSection/OrderPaymentDetails.jsp</code>
<code>mOrderShippingBillingSummaryView</code>	<code>/mobile/ShoppingArea/OrderSection/OrderSummaryDisplay.jsp</code>
<code>mOrderShippingBillingConfirmationView</code>	<code>/mobile/ShoppingArea/OrderSection/OrderConfirmationDisplay.jsp</code>

5.4.4 Design new shopping flow

The analysis in 5.4.3, “Analyze the existing Madisons Mobile Starter Store code” on page 228 shows how control is passed between the pages where we need to inject new pages.

Our analysis also shows that the current mobile checkout pages pass information in the request parameters between pages until the payment information page is shown. Then, the `OrderChangeServicePIAdd` struts action is called. This action is mapped to the `addPaymentInstruction` service in the Order component service module. The `addPaymentInstruction` service allows a client to add payment information, including the billing address.

We will implement our extra pages in a similar manner, passing information about the selected shipping address to the shipping method page, which will then invoke the `OrderChangeServiceShipInfoUpdate` struts action. This action is mapped to the `updateOrderShippingInfo` service. This service allows update of the shipping address, shipping method, and requested shipping date.

Because the `OrderChangeServiceShipInfoUpdate` struts action allows the update of the shipping address, we use this service to update the shipping address and to modify the payment details so that the JSP is not set to a specific shipping address if the shipping mode is not *pick up in store*.

There is, however, a twist in that the shipping instruction page is shown *only* if the check box on the shipping method page for adding shipping instructions is select. We handle this action with JavaScript on the shipping method page by changing the URL parameter to the `OrderChangeServiceShipInfoUpdate` depending on the status of the shipping instruction check box and by updating the shipping instruction on the shipping instruction page. The disadvantage of this approach is that we invoke the service twice for shoppers that specify shipping instructions. The advantage is a slightly simpler flow.

Figure 5-29 illustrates the flow of data for the new checkout pages and how they fit into the existing flow.

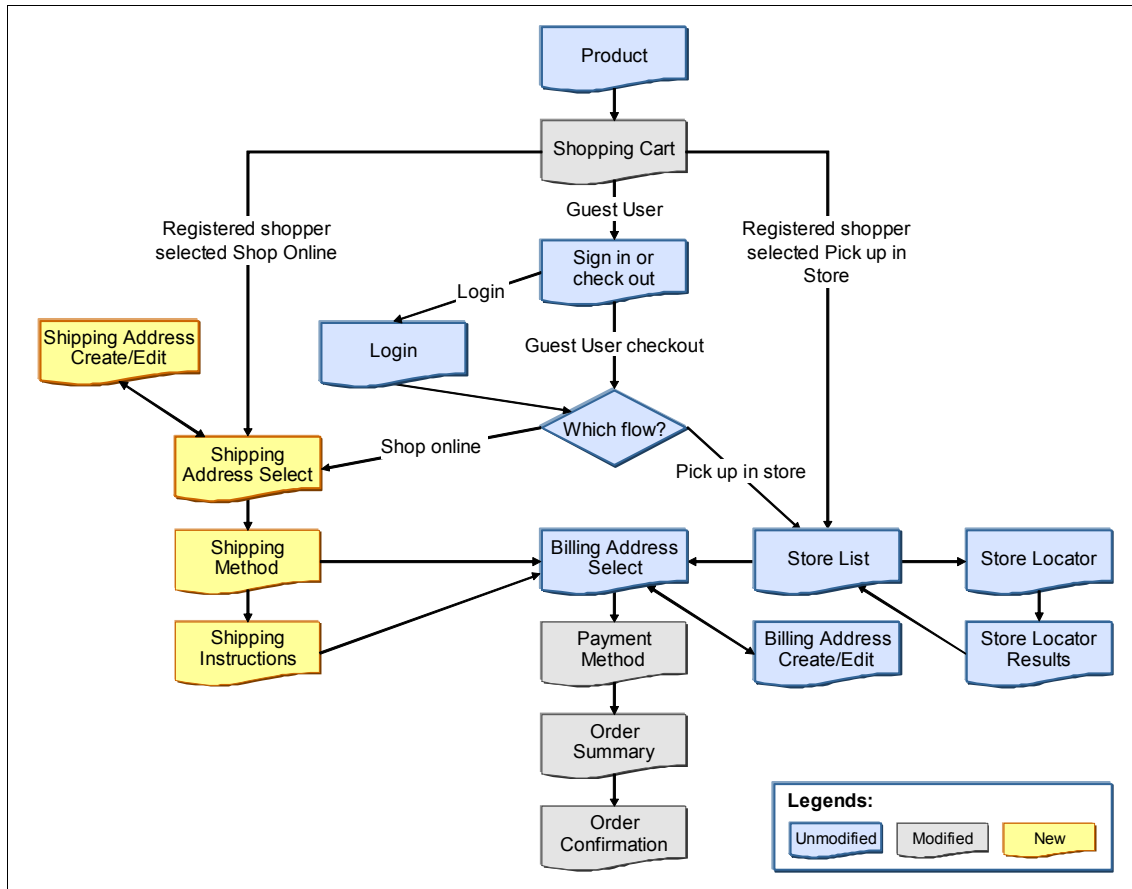


Figure 5-29 Flow chart for the new checkout pages

As the flow chart in Figure 5-29 illustrates, there are several places where the flow of control is determined by the shopper's selections. The recommended approach is to use controller commands to redirect the shopper to the correct page. In our example, for the sake of simplicity, we use JavaScript within the pages to control the flow.

As you might have noticed in the analysis in “Analyze the current checkout pages” on page 229, we did not modify the Store List and Billing Address Select pages.

5.4.5 Create new pages

After analyzing the existing pages in 5.4.3, “Analyze the existing Madisons Mobile Starter Store code” on page 228 and designing the solution in 5.4.4, “Design new shopping flow” on page 239, we are now ready to start implementing the modifications.

We start by creating and testing the new shipping pages. To summarize, we create the four pages outlined in Table 5-8.

Table 5-8 New pages created for selecting and editing shipping addresses

New View Name	New JSP Location
mOrderShippingAddressSelection	/mobile/ShoppingArea/CheckoutSection/OrderShippingAddressSelection.jsp
mOrderShippingDetails	/mobile/ShoppingArea/CheckoutSection/OrderShippingDetails.jsp
mOrderShippingMethodSelection	/mobile/ShoppingArea/CheckoutSection/OrderShippingMethodSelection.jsp
mOrderShippingInstructions	/mobile/ShoppingArea/CheckoutSection/OrderShippingInstructions.jsp

To create the new pages, you need to complete the following steps:

1. Register new pages in the struts configuration.
2. Configure access control policies for new pages.
3. Create the new JSPs.

The following sections guide you through these steps.

Register new pages in the struts configuration

Tip: In this section, we use the Struts editor to add the actions. If you prefer to edit the source code of the `struts-config-ext.xml`, you can skip these steps and use the information from Example 5-15 on page 245 and Example 5-16 on page 246.

You first create the appropriate action mappings and global forwards in the struts configuration for the new pages:

1. Open the development environment by selecting **Start** **Programs** **IBM** **WebSphere** **WebSphere Commerce** **WebSphere Commerce Developer Enterprise**.
2. Switch to the Java EE perspective if it is not already active:
 - a. Select **Window** **Open Perspective** **Other**.
 - b. The Open Perspective window opens. Select **Java EE**, and click **OK**.

3. In the Enterprise Explorer, expand and right-click **Stores** ∅ **WebContent** ∅ **WEB-INF** ∅ **struts-config-ext.xml**.
4. This time, we use the struts configuration file editor to edit the file. From the context menu, select **Open With** ∅ **Other**.
5. The Editor Selection window opens. Select **Struts Configuration File Editor**, and click **OK**.
6. The struts-config-ext.xml file opens in a dedicated editor.
7. For each of the new views listed in Table 5-8 on page 241, create an action mapping and a global forward:
 - a. If you are not already on the Action Mappings tab, click **Action Mappings** from the bottom of the editor window.
 - b. In the Action Mappings area of the window, click **Add**.
 - c. A new entry with the name of /action1 should appear at the bottom of the list of action mappings, as shown in Figure 5-30. Change the name in-line in the list to /mOrderShippingAddressSelection, and press **Enter**.

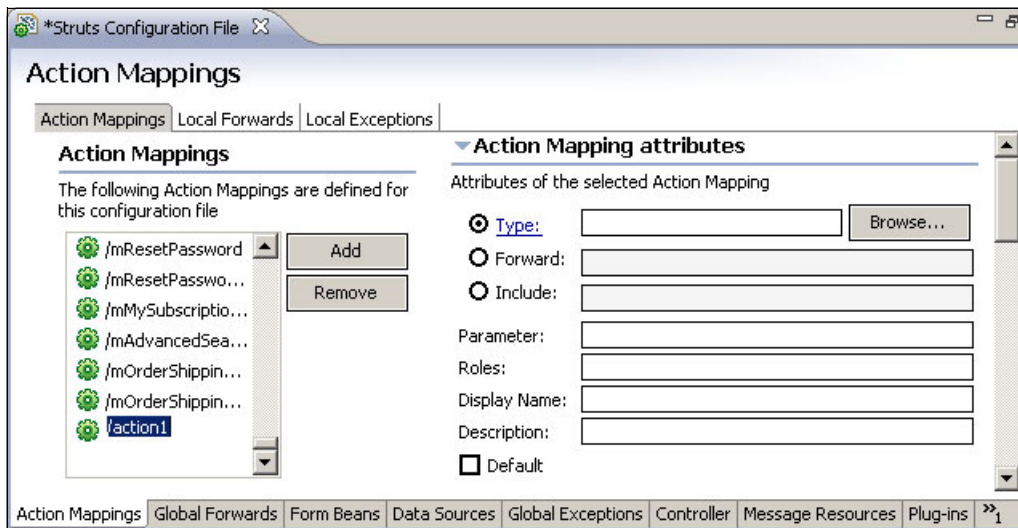


Figure 5-30 New action mapping

- d. Ensure that this action mapping is using the IBM WebSphere Commerce action. In the Action Mapping Attributes section, enter the following in the **Type** field, as shown in Figure 5-31:


```
com.ibm.commerce.struts.BaseAction
```

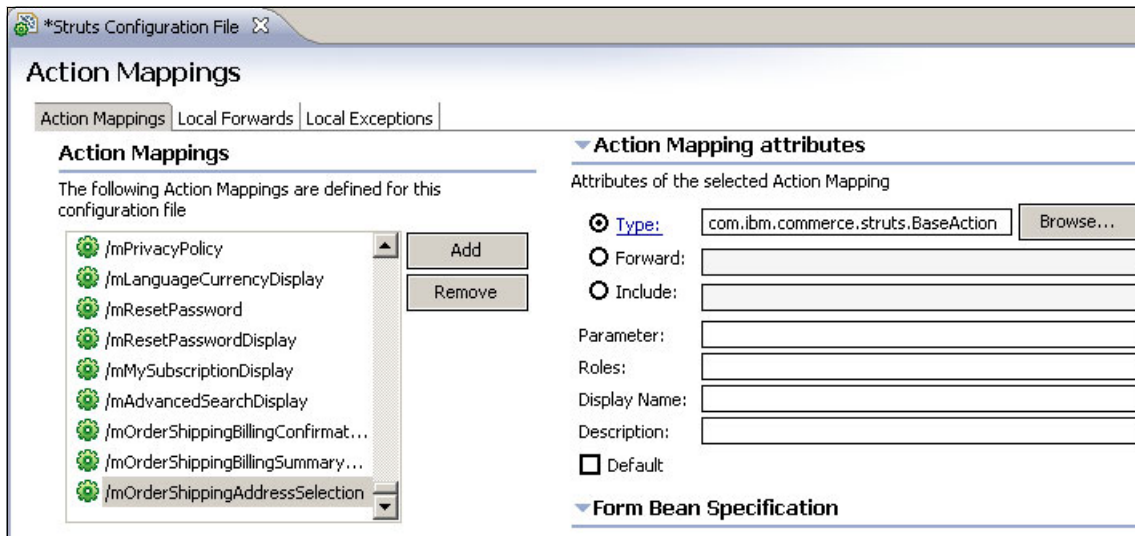



Figure 5-31 Complete action mapping for mOrderShippingAddressSelection

- e. The actions that you are adding relate to sensitive data, such as customer addresses. In addition, you need to ensure that CSRF protection works for the checkout flow. So, you need to configure the actions to require HTTPS by completing the following steps in the Action Mapping Extensions section:
 - i. Click **Add**.
 - ii. A new line is displayed in the table. In the Attribute column, select **property**.
 - iii. In the Key/Property column, enter https.
 - iv. In the Value column, enter 0:1.
 - v. Press Ctrl+S to save the changes.
- f. Create the global forward. Switch to the Global Forwards part of the struts configuration by clicking the Global Forwards tab at the bottom of the editor window.
- g. In the Global Forwards section, click **Add** to create a new forward.
- h. A new entry with the name of success should appear at the bottom of the list of global forwards, as shown in Figure 5-32. Change the name in-line in the list to /mOrderShippingAddressSelection, and press **Enter**.

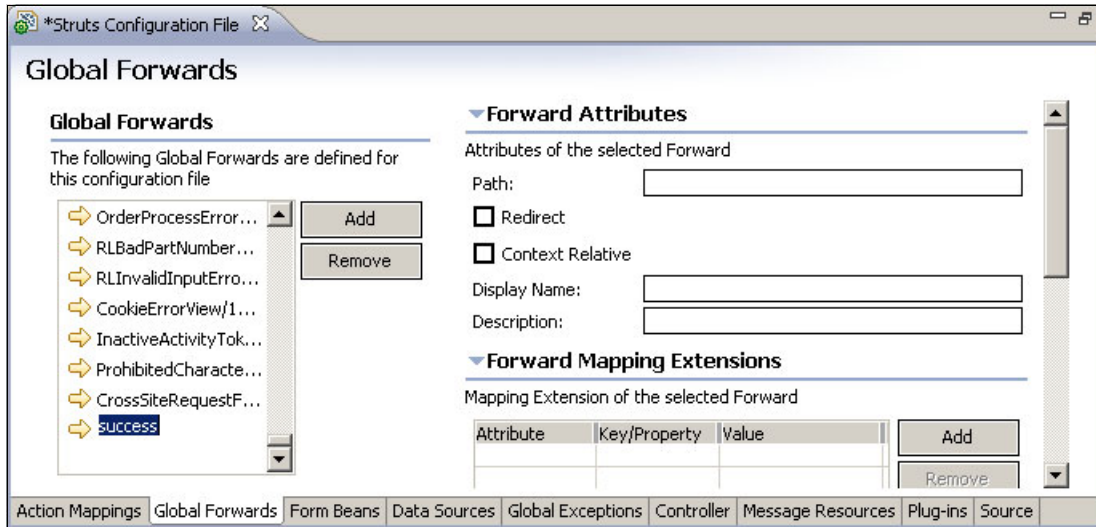


Figure 5-32 New global forward in struts-config-ext.xml

- i. Change the text from success to:


```
mOrderShippingAddressSelection/storeId
```

Where *storeId* is the store ID of your mobile store. Our Madisons Mobile Starter Store had store ID 10051, so we entered the following *storeId* in our example:

```
mOrderShippingAddressSelection/10051
```
- j. In the Path entry fields in Forward Attributes section, enter the name of the JSP from Table 5-8 on page 241. We entered the following name:


```
/mobile/ShoppingArea/CheckoutSection/OrderShippingAddressSelection.jsp
```
- k. In the Class entry field under the Forward Mapping Extensions, enter the following class name:


```
com.ibm.commerce.struts.ECActionForward
```
- l. The new forward mapping should now look like that shown in Figure 5-33. Press Ctrl+S to save your changes.

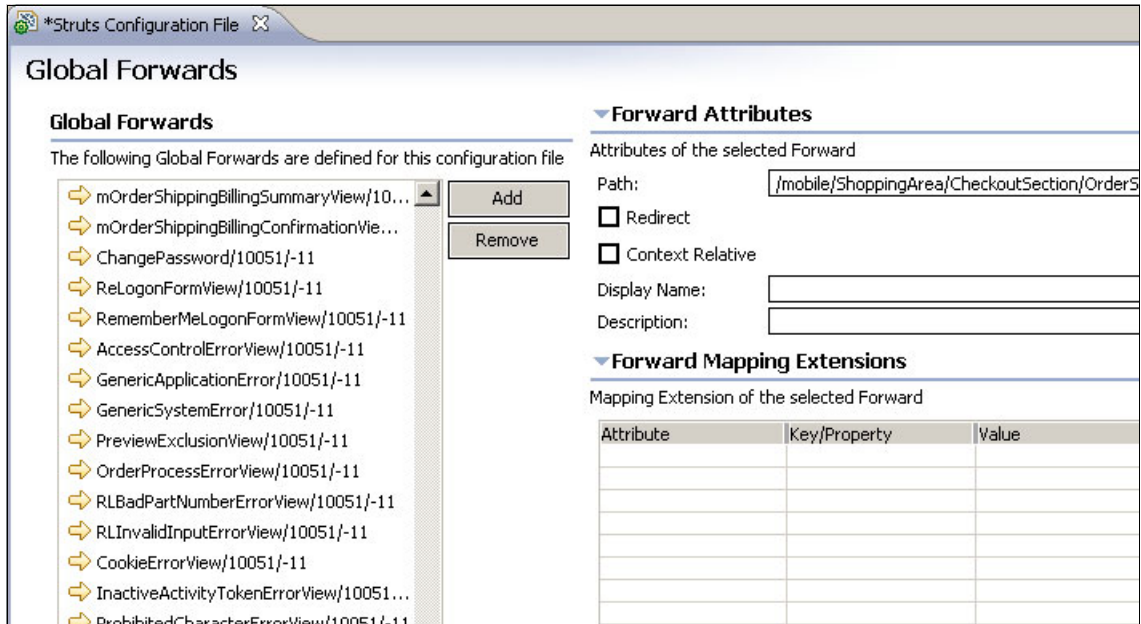


Figure 5-33 The new completed global forward

8. Repeat steps 1 through 7 for the remaining rows in Table 5-8 on page 241. At the end, the source of the struts-config-ext.xml file should contain the action mappings shown in Example 5-15. You can verify this information by clicking the Source tab at the bottom of the editor.

Example 5-15 Action mappings for the new pages


```
<action
  path="/mOrderShippingAddressSelection"
  type="com.ibm.commerce.struts.BaseAction">
    <set-property property="https" value="0:1"/>
</action>
<action
  path="/mOrderShippingDetails"
  type="com.ibm.commerce.struts.BaseAction">
    <set-property property="https" value="0:1"/>
</action>
<action
  path="/mOrderShippingMethodSelection"
  type="com.ibm.commerce.struts.BaseAction">
    <set-property property="https" value="0:1"/>
</action>
<action
```

```
    path="/mOrderShippingInstructions"
    type="com.ibm.commerce.struts.BaseAction">
      <set-property property="https" value="0:1"/>
    </action>
```

Furthermore, the `struts-config-ext.xml` should now contain the global forwards shown in Example 5-16.

Example 5-16 Global forwards for the new pages

```
<forward name="mOrderShippingAddressSelection/10051"
  path="/mobile/ShoppingArea/CheckoutSection/OrderShippingAddressSelection.jsp"
  className="com.ibm.commerce.struts.ECActionForward">
</forward>
<forward name="mOrderShippingDetails/10051"
  path="/mobile/ShoppingArea/CheckoutSection/OrderShippingDetails.jsp"
  className="com.ibm.commerce.struts.ECActionForward">
</forward>
<forward name="mOrderShippingMethodSelection/10051"
  path="/mobile/ShoppingArea/CheckoutSection/OrderShippingMethodSelection.jsp"
  className="com.ibm.commerce.struts.ECActionForward">
</forward>
<forward name="mOrderShippingInstructions/10051"
  path="/mobile/ShoppingArea/CheckoutSection/OrderShippingInstructions.jsp"
  className="com.ibm.commerce.struts.ECActionForward">
</forward>
```

9. Close the `struts-config-ext.xml` editor by selecting **File**  **Close**.

Configure access control policies for new pages

Next, you need to set up the appropriate access control policies before any shopper can access the new pages that you created. IBM WebSphere Commerce uses an explicit access control model in which resources only are accessible to a given group of users if they have been specifically set up to be accessible to that group.¹

The four new pages listed in Table 5-8 on page 241 need the same level of access control as the remaining mobile checkout pages. So, you need to analyze the existing access control policies for one of those pages and model the new pages from that analysis.

¹ The exception to this rule is the access control policy called *SiteAdministratorCanDoEverything*. This policy allows users with the role of Site Administrator to perform any type of action on any resource, regardless of other policies in place for that resource.

Analyze access control policies for mOrderBillingAddressSelection

We used the IBM WebSphere Commerce Organization Administration Console to browse the existing access control policies for the billing address selection page. This page has the view name *mOrderBillingAddressSelection*. To browse the access control policies for mOrderBillingAddressSelection:


1. Open the Organization Administration Console in Microsoft Internet Explorer by visiting the following URL:

`https://localhost:8004/webapp/wcs/orgadmin/servlet/ToolsLogon?XMLFile=buyerconsole.BuyAdminConsoleLogon`

2. Enter the user name and password of the site administration user. We entered the following information:

User name: wcsadmin
Password: wcsadmin

Tip: The default user name and password for the site administration user within the toolkit is wcsadmin/wcsadmin. The first time that you log on using these credentials, you are asked to change the password. After you change the password, click **Change** to continue. Be sure to remember the new password, because you will need it for future steps.

3. In the Organization Administration Console welcome window, select **Access Management**  **Action Groups**. A complete list of action groups opens.


You cannot search for an action group using a specific action name in the Organization Administration Console. So, you have to make an educated guess on the selection of an action group name and then check to see whether the action is included in that action group.

Note: Note that the list of action groups is in alphabetical order.

First look for a specific Madisons Starter Store or Madisons Mobile Starter Store action group. Click **Next** until you get find an action group that starts with *Madisons*. In our example, the first action group that started *Madisons* with was on page eight of the list.

4. Select **MadisonsAllUsersViews**, and click **Show Actions**. The list of actions in the MadisonsAllUsersViews action group is displayed. In our example, we found 11 pages of actions. Scroll through the list one page at a time by clicking **Next**, and look for mOrderBillingAddressSelection.

In our scenario, we did not find mOrderBillingAddressSelection, so we had to look in another action group.

5. Go back to the list of action groups by selecting **Access Management**  **Action Groups**. The complete list of action groups is displayed again.

Because we did not find the action we need in the Madisons-specific action group, we try the general action group for all site views called `AllSiteUsersViews`.

6. Select **AllSiteUsersViews**, and click **Show Actions**.

Tip: The `AllSiteUsersViews` action group should be on the first page. If you do not see it on this page, scroll to the next page until you find this action group.

7. The list of actions in the `AllSiteUsersViews` action group is displayed. In our example, we found 12 pages of actions. Scroll through the list one page at a time by clicking **Next**, and look for `mOrderBillingAddressSelection`.

In our example, we found `mOrderBillingAddressSelection` on page 11, along with the remaining mobile views, which tells us that the mobile views are in the action group `AllSiteUsersViews`.

Note: Although the list of actions is sorted alphabetically, the sorting order is the standard ASCII order, which places all the lowercase letters after the uppercase letters. Thus, the actions that begin with a lowercase *m* show at the end of the list, *not* with the actions that begin with an uppercase *M*.

Now that you have identified the action group, you need to determine the organization that owns this action group. You will need this information to load the new actions into the action group. You can obtain this information using one of the following methods:

- ▶ Search for the definition in the XML files in `WC_Home\xml\policies\xml`.
- ▶ Look up the organization name in the database into which you load the policies.

In our scenario, we use the first option; however, for completeness, Example 5-17 provides sample SQL that you can use to determine the organization name and ID for the `AllSiteUsersViews` action group.

Example 5-17 SQL for determining the organization name and ID for an action group

```
SELECT GROUPNAME, ORGENTITYNAME, ORGENTITY_ID
FROM AACTGRP, ORGENTITY
WHERE GROUPNAME = 'AllSiteUsersViews'
AND AACTGRP.MEMBER_ID = ORGENTITY.ORGENTITY_ID
```

To determine the owning organization for a standard IBM WebSphere Commerce action group through the standard policy XML files:

1. Open the following file in your favorite text editor.

WC_Home\xml\policies\xml\defaultAccessControlPolicies.xml

We opened Notepad by selecting **Start** ⌵ **Programs** ⌵ **Accessories** ⌵ **Notepad**.

2. Search for the following text:

```
<ActionGroup Name="AllSiteUsersViews"
```

We used Ctrl+F to use the search function in Notepad

3. The snippet shown in Example 5-18 is displayed. The highlighted line marks the searched for text.

Example 5-18 Snippet from defaultAccessControlPolicies.xml showing the action group

```
<ActionGroup Name="UserRoleAssign" OwnerID="RootOrganization" >
  <ActionGroupAction Name="com.ibm.commerce.usermanagement.commands.MemberRoleAssignCmd-User"/>
  <ActionGroupAction Name="com.ibm.commerce.usermanagement.commands.MemberRoleUnassignCmd-User"/>
</ActionGroup>

<!-- Group all site views together into one action group -->
<ActionGroup Name="AllSiteUsersViews" OwnerID="RootOrganization">
  <ActionGroupAction Name="AdminConHome"/>
  <ActionGroupAction Name="AdminConLaunched"/>
  <ActionGroupAction Name="AdminConSiteStoreSelection"/>
  <ActionGroupAction Name="ButtonView"/>
</ActionGroup>
```

Example 5-18 shows that AllSiteUsersViews is owned by Root Organization. You can copy the <ActionGroup> line from this file and use it in the access control policy file as described in the next section.

Create access control policies for the new views

Because mOrderBillingAddressSelection is located in the action group AllSiteUsersViews, you must also place the four new views as actions in this action group to allow any user in the site to access the new views. To create access control policies for the new views:

1. Open your favorite text editor. We opened Notepad by selecting **Start** ⌵ **Programs** ⌵ **Accessories** ⌵ **Notepad**.
2. Enter the XML document shown in Example 5-19 into the text editor.

The XML in Example 5-19 should be straightforward to read. Basically, it defines the four new actions using <Action> XML nodes and then associates these actions to the AllSiteUsersViews action group. You can determine the reference to RootOrganization in the <ActionGroup> node from the bootstrap default access control file:

WC_Home/xml/policies/xml/defaultAccessControlPolicies.xml

Another approach is to check the contents of the ACPOLICY table in the IBM WebSphere Commerce instance database.

We obtained the value from the bootstrap default access control policy file.

Example 5-19 Access control policies for the custom views

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no" ?>

<!DOCTYPE Policies SYSTEM "../dtd/accesscontrolpolicies.dtd">
<Policies>

    <!-- BEGIN: Define the actions -->
    <Action Name="mOrderShippingAddressSelection"
        CommandName="mOrderShippingAddressSelection" />

    <Action Name="mOrderShippingDetails"
        CommandName="mOrderShippingDetails" />

    <Action Name="mOrderShippingMethodSelection"
        CommandName="mOrderShippingMethodSelection" />

    <Action Name="mOrderShippingInstructions"
        CommandName="mOrderShippingInstructions" />
    <!-- END: Define the actions -->

    <!-- Map the actions to the AllSiteUsersViews action group -->
    <ActionGroup Name="AllSiteUsersViews" OwnerID="RootOrganization">
        <ActionGroupAction Name="mOrderShippingAddressSelection"/>
        <ActionGroupAction Name="mOrderShippingDetails"/>
        <ActionGroupAction Name="mOrderShippingMethodSelection"/>
        <ActionGroupAction Name="mOrderShippingInstructions"/>
    </ActionGroup>

</Policies>
```

3. Save the file as `WC_Home\xml\policies\xml\mobile-shipping-acp.xml`. In our example, we saved the file as:

`C:\IBM\WCDE_ENT70\xml\policies\xml\mobile-shipping-acp.xml`

4. Close the text editor.




Load the access control policies for the new views

Because you have now created the access control policy file and placed it in the location to be picked up by IBM WebSphere Commerce, you can load these new views into the development database.

Important: The default database provider for IBM WebSphere Commerce V7 Developer Edition is IBM Cloudscape. IBM Cloudscape allows only one connection to the database at a time. As such, it is not possible to load the access control policies while the server is running.

If you use IBM Cloudscape for the database, you need to stop the server while the access control policies are loaded and then restart the server afterwards.

To load the custom access control policies:

1. Open a command line window by selecting **Start**  **Programs**  **Accessories**  **Command Prompt**.
2. Change to the `WC_Home\bin` directory, where `WC_Home` is the installation directory of IBM WebSphere Commerce V7 Developer Edition. For example, we entered the following command:

```
cd \IBM\WCDE_ENT70\bin
```

3. Run the following command to transform the XML and load the new policies:

```
acpload mobile-shipping-acp.xml
```

Note: If you are not using IBM Cloudscape as the database manager, you need to specify connection parameters on the **acpload** command line. Run **acpload** without any parameters to learn the syntax of the command.

4. The **acpload** command runs for a while. Example 5-20 shows the expected output.

Example 5-20 Expected output from acpload

```
C:\IBM\WCDE_ENT70\bin>acpload mobile-shipping-acp.xml
Running XMLTransform...
Running Id Resolver...
Running MassLoader...
```

Examine `acpload.log` to ensure that everything completed successfully.

```
C:\IBM\WCDE_ENT70\bin>
```

5. Because the output from *acpload* might not notify you of errors, you need to inspect the `acpload.log` file for errors, as stated in the console output from **acpload**.

The `acpload.log` file is located in the `WC_Home\logs` directory. In our example, the file was located in the following directory:

```
C:\IBM\WCDE_ENT70\logs
```

Example 5-21 shows a sample of the output in the `acpload.log` file.

Example 5-21 Output in `acpload.log` from a successful load

```
Running XMLTransform...
Running Id Resolver...
Running MassLoader...
```

If you are using IBM Cloudscape and have not stopped the server, the `acpload.log` log will contain an error similar to the one shown in Example 5-22. If you receive this error, stop the application that is using the database (most likely the test server), and retry step 3 on page 251.

Example 5-22 Error message from `acpload` if the IBM Cloudscape database is in use

```
Running XMLTransform...
Running Id Resolver...
Running MassLoader...
Failed to start database 'C:\IBM\WCDE_E~1\db\mall', see the next
exception for details.
```

Another method to verify that the load process has completed successfully is to examine the transformed and ID resolved XML files. These files are located in the same directory as the `mobile-shipping-acp.xml` file, the `WC_Home\xml\policies\xml` directory.

In our example, the C:\IBM\WCDE_ENT70 directory included the following files:

– mobile-shipping-acp_xmltrans.xml

In our example, this file had the content shown in Example 5-23 (reformatted here for readability). When checking for errors, it is important to verify the following information:

- All attributes must have values. For example, no attributes are specified with empty strings as the value.
- All actions must have corresponding <acaction> and <acactactgp> nodes in the XML document.
- There must be one <acactgrp> node that corresponds to the <ActionGroup> node shown in Example 5-19 on page 250.

Example 5-23 The transformed access control policy file

```
<?xml version="1.0" encoding="UTF-8"?><!DOCTYPE import SYSTEM
"../../schema/xml/wcs.dtd"><import>

  <acaction ACACTION_ID="@mOrderShippingAddressSelection"
    ACTION="mOrderShippingAddressSelection"/>
  <acaction ACACTION_ID="@mOrderShippingDetails"
    ACTION="mOrderShippingDetails"/>
  <acaction ACACTION_ID="@mOrderShippingMethodSelection"
    ACTION="mOrderShippingMethodSelection"/>
  <acaction ACACTION_ID="@mOrderShippingInstructions"
    ACTION="mOrderShippingInstructions"/>

  <acactgrp ACACTGRP_ID="@AllSiteUsersViews"
    GROUPNAME="AllSiteUsersViews"
    MEMBER_ID="-2001"/>

  <acactactgp ACACTGRP_ID="@AllSiteUsersViews"
    ACACTION_ID="@mOrderShippingAddressSelection"/>
  <acactactgp ACACTGRP_ID="@AllSiteUsersViews"
    ACACTION_ID="@mOrderShippingDetails"/>
  <acactactgp ACACTGRP_ID="@AllSiteUsersViews"
    ACACTION_ID="@mOrderShippingMethodSelection"/>
  <acactactgp ACACTGRP_ID="@AllSiteUsersViews"
    ACACTION_ID="@mOrderShippingInstructions"/>

</import>
```

– mobile-shipping-acp_idres.xml

In our example, this file had the content shown in Example 5-24 (reformatted and abbreviated here for readability). When checking for errors, it is important to note the following information:

- A very long DTD preamble, which defines XML nodes for all standard IBM WebSphere Commerce V7 tables, must be present.
- All the XML nodes from the mobile-shipping-acp_xmltrans.xml file must be present, although with different attributes.
- Apart from the ACTION attributes of the <acaction> nodes, all attributes must have numeric values.

Example 5-24 The ID resolved access control policy file

```
<?xml version="1.0"encoding="UTF-8"?>
<!-- Name:
C:\IBM\WCDE_E~1\xml\policies\xml\mobile-shipping-acp_idres.xml
Description: This file contain data generated by IBM utility.
-->

<!DOCTYPE import [
<!ELEMENT import ((acacgpdesc|acactactgp|acactdesc|acactgrp|acaction
|acattr|acattrdesc|acccmdgrp|acccmdtype|accustexc|acclogmain|acclog
sub|ac

...many lines of DTD declarations removed...

<!ELEMENT wusrtrvw EMPTY>
<!-- ATTLIST wusrtrvw
STOREENT_ID CDATA #REQUIRED
TOTALVISITS CDATA #REQUIRED
TOTALSESSIONS CDATA #REQUIRED
DISTINCTUSERS CDATA #REQUIRED
>
]>

<import>
  <acaction
    ACACTION_ID="13151"
    ACTION="mOrderShippingAddressSelection"
  />
  <acaction
    ACACTION_ID="13152"
    ACTION="mOrderShippingDetails"
  />
```

```

    <acaction
      ACACTION_ID="13153"
      ACTION="mOrderShippingMethodSelection"
    />
    <acaction
      ACACTION_ID="13154"
      ACTION="mOrderShippingInstructions"
    />
    <acactactgp
      ACACTGRP_ID="10196"
      ACACTION_ID="13151"
    />
    <acactactgp
      ACACTGRP_ID="10196"
      ACACTION_ID="13152"
    />
    <acactactgp
      ACACTGRP_ID="10196"
      ACACTION_ID="13153"
    />
    <acactactgp
      ACACTGRP_ID="10196"
      ACACTION_ID="13154"
    />
  </import>

```

Create the new JSPs

Now that you have loaded the access control policies for the new views, you can create the JSPs that render these views. As listed in Table 5-8 on page 241, you need to create the following JSPs:

- ▶ OrderShippingAddressSelection.jsp
- ▶ OrderShippingDetails.jsp
- ▶ OrderShippingMethodSelection.jsp
- ▶ OrderShippingInstructions.jsp














All of these pages will reside within the mobile checkout directory for the Madisons Mobile Starter Store, which in IBM WebSphere Commerce V7 Developer Edition is the following directory:

WC_Home\workspace\Stores\WebContent\Madisons\mobile\ShoppingArea\
CheckoutSection

Create the OrderShippingAddressSelection.jsp file

You create a new OrderShippingAddressSelection.jsp that displays all the current shipping addresses, which allows the shopper to select an address to which to ship the merchandise. An alternative method to simplify changes to both pages is to create a generic address selection JSP and then pass that JSP to the address type as a parameter. However, we chose to introduce another JSP to minimize the number of edited starter store pages in this example.

To create the shipping address selection JSP:

1. Open the development environment by selecting **Start**  **Programs**  **IBM**  **WebSphere**  **WebSphere Commerce**  **WebSphere Commerce Developer Enterprise**.
2. Switch to the Java EE perspective if it is not already active:
 - a. Select **Window**  **Open Perspective**  **Other**.
 - b. The Open Perspective window opens. Select **Java EE**, and click **OK**.
3. In the Enterprise Explorer, expand and right-click **Stores**  **WebContent**  **Madisons**  **mobile**  **ShoppingArea**  **CheckoutSection**.
4. In the context menu that opens, select **New**  **File**.
5. The New File dialog box opens. Enter OrderShippingAddressSelection.jsp in the File name field, and click **Finish**.
6. The JSP file editor opens in the source tab and shows a blank page. Enter the skeleton JSP file shown in Example 5-25 into the page, and press Ctrl+S to save the file. Do *not* close the file yet.

The code in Example 5-25 provides the skeleton in which you can add the code for the page. Run by itself, this example code produces only the header and footer for the page. You can insert the remaining content at the JSP comment place-holders.

Example 5-25 Skeleton code for OrderShippingAddressSelection.jsp

```
<%--
    *****
    * This JSP displays all existing shipping addresses, and allows the user to
    * select the address for checkout
    *****
--%>
<!-- BEGIN OrderShippingAddressSelection.jsp -->
<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="http://commerce.ibm.com/base" prefix="wcbase" %>
<%@ taglib uri="http://commerce.ibm.com/foundation" prefix="wcf" %>
```

```

<%@ include file="../../include/parameters.jspf" %>
<%@ include file="../../include/JSTLEnvironmentSetup.jspf" %>
<%-- 1. Initialization --%>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML Basic 1.1//EN"
    "http://www.w3.org/TR/xhtml1-basic/xhtml1-basic11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="${shortLocale}"
    xml:lang="${shortLocale}">
    <head>
        <title>
            <fmt:message key="SHIPPING_ADDRESS_SELECT_TITLE" bundle="${storeText}"/>
            - <c:out value="${storeName}"/>
        </title>
        <meta http-equiv="content-type" content="application/xhtml+xml" />
        <meta http-equiv="cache-control" content="max-age=300" />
        <meta name="viewport"
            content="width=device-width, initial-scale=1.0, user-scalable=no" />
        <link rel="stylesheet" type="text/css" href="${cssPath}" />
        <%-- 4. Javascript validation --%>
    </head>
    <body>
        <div id="wrapper">
            <%@ include file="../../include/HeaderDisplay.jspf" %>
            <%@ include file="../../include/BreadCrumbTrailDisplay.jspf" %>
            <div id="address_list" class="content_box">
                <%-- 2. Main content --%>
            </div>
            <%@ include file="../../include/FooterDisplay.jspf" %>
        </div>
    </body>
</html>
<!-- END OrderShippingAddressSelection.jsp -->

```

7. In the JSP file, locate the following line:

```
<%-- 1. Initialization --%>
```

Add the code shown in Example 5-26 after this line. This code executes the following actions:

- Initializes some variables used in the breadcrumb JSP fragment for rendering the correct breadcrumb path.
- Sets up URLs used in the main content for transitioning to the next page or removing an address.

- Invokes the `findCurrentShoppingCart` and `findCurrentPerson` services to retrieve data that is needed in rendering the page.
- Determines the number and nature of addresses for the current user by inspecting the address in the shopper's address book.

Important: Technically, this list of addresses can contain addresses that are not valid as shipping addresses. IBM WebSphere Commerce operates with the following address types, as defined by the `ADDRESSTYPE` column of the `ADDRESS` table:

- ▶ Billing-only address (`ADDRESSTYPE='B'`)
- ▶ Shipping-only address (`ADDRESSTYPE='S'`)
- ▶ General address (`ADDRESSTYPE='SB'` used as both the billing and shipping address)

By default, IBM WebSphere Commerce creates a general address. We use this type of address and assume that all addresses are valid as shipping addresses. A more complete implementation can filter out the billing-only addresses from the list.

Example 5-26 Initialization code for `OrderShippingAddressSelection.jsp`

```
<%-- Required variables for breadcrumb support --%>
<c:set var="shoppingcartPageGroup" value="true" scope="request"/>
<c:set var="shippingSelectionPage" value="true" scope="request"/>

<%-- URL for next page --%>
<wcf:url var="OrderShippingDetailsURL" value="mOrderShippingDetails">
  <wcf:param name="langId" value="{langId}" />
  <wcf:param name="storeId" value="{WCPParam.storeId}" />
  <wcf:param name="catalogId" value="{WCPParam.catalogId}" />
  <wcf:param name="orderId" value="{WCPParam.orderId}" />
</wcf:url>
<%-- URL for removing an address --%>
<wcf:url var="AddressDeleteURL" value="PersonChangeServiceAddressDelete">
  <wcf:param name="langId" value="{langId}" />
  <wcf:param name="storeId" value="{WCPParam.storeId}" />
  <wcf:param name="catalogId" value="{WCPParam.catalogId}" />
  <wcf:param name="URL" value="mOrderShippingAddressSelection" />
</wcf:url>

<%-- Retrieve details about the current order
      (mainly to get the current shipping address) --%>
<wcf:getData type="com.ibm.commerce.order.facade.datatypes.OrderType"
```



```

    var="order" expressionBuilder="findCurrentShoppingCart" scope="request">
    <wcf:param name="accessProfile" value="IBM_Details" />
</wcf:getData>

<%-- Retrieve data for the current user - we do this to get the address book
and the self address --%>
<wcf:getData type="com.ibm.commerce.member.facade.datatypes.PersonType"
    var="person" expressionBuilder="findCurrentPerson">
    <wcf:param name="accessProfile" value="IBM_All" />
</wcf:getData>

<%-- get the current shipping address ID --%>
<c:set var="shipInfo" value="${order.orderItem[0].orderItemShippingInfo}" />
<c:set var="currentShip"
    value="${shipInfo.shippingAddress.contactInfoIdentifier.uniqueID}" />

<%-- determine which addresses the current shopper has --%>
<c:set var="hasValidAddresses" value="false"/>
<c:set var="hasSelfAddress" value="${!empty person.contactInfo}" />
<c:set var="addressBook" value="${person.addressBook.contact}" />
<c:set var="numAddresses" value="${fn:length(addressBook)}" />
<c:if test="${(numAddresses > 0 || hasSelfAddress) && !hasValidAddresses}">
    <c:set var="hasValidAddresses" value="true"/>
</c:if>

```

8. In the JSP file, locate the following line:

```
<%-- 2. Main content --%>
```

Add the code shown in Example 5-27 after this line. This code executes the following actions:

- Displays the page title.
- Loops through the shopper's addresses (with one extra loop iteration if the shopper has a self address) and completes the following steps:
 - Determines whether the current iteration displays the self address (handled in a special first iteration).
 - Determines whether the current address corresponds to the currently selected shipping address for the order.
 - Displays the address. (The code is added in a later step.)
- Adds a link to create a new address (redirect to mOrderShippingDetails).
- Adds a button to submit the selected shipping address to mOrderShippingMethodSelection. The submission is done through the Java Script function checkAddress, which is added in the next step.

```
<div class="heading_container_with_underline">
  <h2><fmt:message key="YOUR_SHIPPING_ADDRESSES" bundle="${storeText}" /></h2>
  <div class="clear_float"></div>
</div>
<p class="paragraph_blurb">
  <fmt:message key="SHIPPING_ADDRESS_SELECT" bundle="${storeText}" />
</p>
<c:choose>
  <c:when test="${hasValidAddresses}">
    <form name="shippingAddressForm">
      <!-- The address book will not contain the self address, so we'll need to
           jump through a few hoops to include this in the loop... --%>
      <!-- first determine if there is a self address to include --%>
      <c:choose>
        <c:when test="${hasSelfAddress}">
          <c:set var="beginIndex" value="0" />
        </c:when>
        <c:otherwise>
          <c:set var="beginIndex" value="1" />
        </c:otherwise>
      </c:choose>
      <!-- then go through all addresses (including the self address) --%>
      <c:forEach var="index" begin="${beginIndex}" end="${numAddresses}">

        <!-- determine if this is the self address or a regular entry --%>
        <c:choose>
          <c:when test="${index == 0}">
            <c:set var="contact" value="${person.contactInfo}" />
          </c:when>
          <c:otherwise>
            <c:set var="contact" value="${addressBook[index-1]}" />
          </c:otherwise>
        </c:choose>
        <c:set var="contactId" value="${contact.contactInfoIdentifier}" />
        <c:set var="addressId" value="${contactId.uniqueID}" />
        <c:choose>
          <c:when test="${currentShip == addressId}">
            <c:set var="optionChecked" value="checked='checked'"/>
          </c:when>
          <c:otherwise>
            <c:set var="optionChecked" value=""/>
          </c:otherwise>
        </c:choose>
      </c:forEach>
    </form>
  </c:when>
  <c:otherwise>
    <!-- No valid addresses --%>
  </c:otherwise>
</c:choose>
```

```

        <ul class="entry">
            <!-- 3. Address Display -->
        </ul>
    </c:forEach>
</form>
</c:when>
<c:otherwise>
    <!-- There are no addresses found.
         Redirect to the address details page to create an address. -->
    <script type="text/javascript">
        window.location.href='<c:out value="\${OrderShippingDetailsURL}" />';
    </script>
</c:otherwise>
</c:choose>
<div>
    <fmt:message key="SHIPPING_ADDRESS_CREATE" bundle="\${storeText}" />
    <p class="paragraph_blurb">
        <span class="bullet">&#187; </span>
        <a href="\${OrderShippingDetailsURL}"
            title="<fmt:message key="CREATE_NEW_ADDRESS"
                bundle="\${storeText}" />"
            <fmt:message key="CREATE_NEW_ADDRESS" bundle="\${storeText}" />
        </a>
    </p>
</div>
<form id="your_store_list_buttons">
    <input type="button" id="continue_checkout" name="continue_checkout"
        value="<fmt:message key="CONTINUE_CHECKOUT" bundle="\${storeText}" />"
        class="input_button_float" onclick="checkAddress();" />
</form>
<form id="continue_checkout_form" action="mOrderShippingMethodSelection">
    <input type="hidden" name="langId" value="\${langId}" />
    <input type="hidden" name="storeId" value="\${WParam.storeId}" />
    <input type="hidden" name="catalogId" value="\${WParam.catalogId}" />
    <input type="hidden" name="addressId" value="" id="shippingAddressId" />
</form>

```

9. In the JSP file, locate the following line:

```
<%-- 3. Address Display --%>
```

Add the code shown in Example 5-28 after this line. This code executes the following actions:

- Determines the language-specific name for the country listed in the address.
- Determines the language-specific name for the state or province listed in the address.
- Displays the various parts of the address.
- Displays links to edit and delete the address. (Deletes the address only if the address is not a self-address.)

Example 5-28 Code to display one shipping address for OrderShippingAddressSelection.jsp

```
<c:set var="countryDisplayName" value="${contact.address.country}"/>
<c:set var="stateDisplayName" value="${contact.address.stateOrProvinceName}"/>
<wcbase:useBean id="countryBean"
    classname="com.ibm.commerce.user.beans.CountryStateListDataBean">
    <c:set target="${countryBean}" property="countryCode"
        value="${contact.address.country}"/>
</wcbase:useBean>
<c:forEach var="country" items="${countryBean.countries}">
    <c:if test="${!empty country.code && country.code == contact.address.country}">
        <c:set var="countryDisplayName" value="${country.displayName}"/>
    </c:if>

    <c:if test="${!empty country.states}">
        <c:forEach var="state" items="${country.states}" varStatus="counter">
            <c:if test="${!empty state.code &&
                state.code == contact.address.stateOrProvinceName}">
                <c:set var="stateDisplayName" value="${state.displayName}"/>
            </c:if>
        </c:forEach>
    </c:if>
</c:forEach>
<li>
    <div class="radio_container">
        <input type="radio" id="shipping_address_selection_${addressId}"
            name="shipping_address_selection"
            value="<c:out value="${addressId}"/>" ${optionChecked} />
        <label for="shipping_address_selection_${addressId}">
            <span class="bold">
                ${contactId.externalIdentifier.contactInfoNickName}
```

```

        </span>
    </label>
</div>
</li>
<li class="align_with_radio">
    <c:out value="\${contact.contactName.firstName}"/>
    <c:out value="\${contact.contactName.lastName}"/>
</li>
<li class="align_with_radio">
    <c:out value="\${contact.address.addressLine[0]}/>
    <c:out value="\${contact.address.addressLine[1]}/>
</li>
<li class="align_with_radio">
    <c:out value="\${contact.address.city}"/> <c:out value="\${stateDisplayName}"/>
</li>
<li class="align_with_radio">
    <c:out value="\${countryDisplayName}"/>
    <c:out value="\${contact.address.postalCode}"/>
</li>
<li class="align_with_radio"><c:out value="\${contact.telephone1.value}"/></li>
<li class="align_with_radio"><c:out value="\${contact.emailAddress1.value}"/></li>
<li class="align_with_radio">
    <span class="bullet">&#187; </span>
    <a href="\${OrderShippingDetailsURL}&addressId=\${addressId}">
        <fmt:message key="MO_EDIT" bundle="\${storeText}"/>
    </a>
</li>

<c:if test="\${person.contactInfo.contactInfoIdentifier.uniqueID != addressId}">
    <li class="align_with_radio">
        <span class="bullet">&#187; </span>
        <a href="\${AddressDeleteURL}&addressId=\${addressId}">
            <fmt:message key="MSTLST_REMOVE_STORE" bundle="\${storeText}"/>
        </a>
    </li>
</c:if>

```

10. In the JSP, locate the following line:

```
<%-- 4. Javascript validation --%>
```

Add the code shown in Example 5-29 after this line. This code submits the form only if the shopper has actually selected one of the addresses in the form.







Example 5-29 Form validation for OrderShippingAddressSelection.jsp

```
<script type="text/javascript">
//
function checkAddress() {
    var addressId = null;
    if(typeof(document.shippingAddressForm) != "undefined") {
        var addressSelect = document.shippingAddressForm.shipping_address_selection;
        if(typeof(addressSelect.length) != "undefined") {
            // Multiple addresses are available
            for(var i = 0; i &lt; addressSelect.length; i++) {
                if(addressSelect[i].checked) {
                    addressId = addressSelect[i].value;
                    break;
                }
            }
        }
        else {
            // Only one address is available
            addressId = addressSelect.value;
        }
        if(addressId != null) {
            document.getElementById("shippingAddressId").value = addressId;
            document.getElementById("continue_checkout_form").submit();
        }
    }
}
//]]&gt;
&lt;/script&gt;</pre><hr/></div><div data-bbox="264 740 524 759" data-label="Text"><p>11. Save the file by pressing Ctrl+S.</p></div><div data-bbox="264 774 847 830" data-label="Text"><p>The OrderShippingAddressSelection.jsp file is now complete. Later, after you test the page, you can define the resource bundles for the page, as described in “Add localized texts” on page 284.</p></div><div data-bbox="146 942 617 961" data-label="Page-Footer"><p><b>264</b> Building Multichannel Applications with WebSphere Commerce</p></div>
```

Create the OrderShippingDetails.jsp file

Next, you create a new `OrderShippingDetails.jsp` file that is used for creating and editing shipping addresses. Because the shipping address details page is very similar to the existing billing address details, we base the `OrderShippingDetails.jsp` file on the corresponding billing address page. As with the shipping address selection JSP, an alternative approach that simplifies changes to both pages is to create a generic address edit JSP and then pass that JSP to the address type as a parameter. However, in our scenario, we chose to introduce this new JSP to simplify the work needed in this example.

To create the shipping address, edit the JSP based on the billing address edit JSP:

1. In the Enterprise Explorer of the Java EE perspective, expand and right-click **Stores**  **WebContent**  **Madisons**  **mobile**  **ShoppingArea**  **CheckoutSection**  **OrderBillingDetails.jsp**.
2. In the context menu that opens, select **Copy**.
3. Right-click the **CheckoutSection** folder, and select **Paste**.
4. The Name Conflict dialog box opens. Enter the following file name in the entry field, and click **OK**:
`OrderShippingDetails.jsp`
5. The new `OrderShippingDetails.jsp` file is displayed. To open the file in an editor, double-click the file name.
6. The JSP editor opens. If the file is shown in the Design, Split, or Preview tabs, you can switch to the Source tab by clicking **Source** at the bottom of the window.
7. Press Ctrl+F to open the Find/Replace window.
8. In the Find/Replace window, enter the following information:

Find:	billing
Replace with:	shipping

Select **All** for the Scope, and ensure that the “Case sensitive” and “Wrap search” options are selected. All other options should be cleared.

Click **Replace All**.

Important: Enter the search terms *exactly* as we show here, including the case of the letters.

The number of items replaced shows at the bottom of the Find/Replace window. In our example, we saw the following message:

6 matches replaced

9. Repeat the search and replace process using the following terms (notice the uppercase first letter of each term):

Find: Billing
Replace with: Shipping

Keep all other options as in step 8.

Again, the number of matches shows. In our example, we saw the following message:

4 matches replaced

10. Repeat the search and replace process using all uppercase spellings:

Find: BILLING
Replace with: SHIPPING

Keep all other options as in step 8.

Again, the number of matches is shown. In our example, we saw the following message:

2 matches replaced

11. Click **Close** to close the Find/Replace window, and save the file by pressing Ctrl+S.

The `OrderShippingDetails.jsp` file is now complete. Next, you define the resource bundles for the page as described in “Add localized texts” on page 284.

Create the OrderShippingMethodSelection.jsp file

In our scenario, we create the new `OrderShippingMethodSelection.jsp` from scratch. Refer to Figure 5-24 on page 226 for the finished look of this page. To create the file:

1. Open the development environment by selecting **Start** ⌵ **Programs** ⌵ **IBM** ⌵ **WebSphere** ⌵ **WebSphere Commerce** ⌵ **WebSphere Commerce Developer Enterprise**.
2. Switch to the Java EE perspective if it is not already active:
 - a. Select **Window** ⌵ **Open Perspective** ⌵ **Other**.
 - b. The Open Perspective window opens. Select **Java EE**, and click **OK**.
3. In the Enterprise Explorer, expand and right-click **Stores** ⌵ **WebContent** ⌵ **Madisons** ⌵ **mobile** ⌵ **ShoppingArea** ⌵ **CheckoutSection**.
4. In the context menu that opens, select **New** ⌵ **File**.

5. The New File dialog box opens. Enter `OrderShippingMethodSelection.jsp` in the File name field, as shown in Figure 5-34, and click **Finish**.

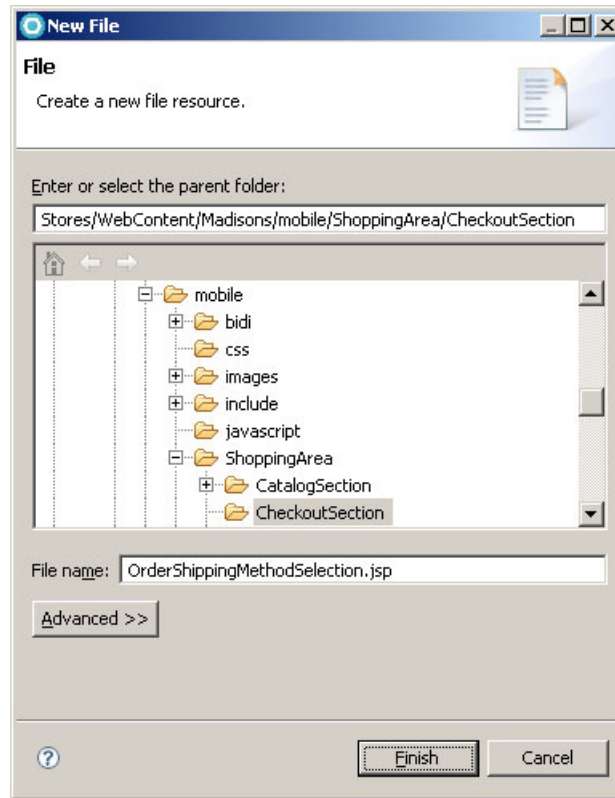


Figure 5-34 Creating `OrderShippingMethodSelection.jsp` using the New File wizard

6. The JSP file editor opens in the source tab and shows a blank page. Enter the skeleton JSP file shown in Example 5-30, and press `Ctrl+S` to save the file. Do *not* close the file yet.

Example 5-30 Skeleton for the `OrderShippingMethodSelection.jsp` file

```
<%--
*****
* This JSP displays the available shipping methods and delivery date, and allows
* the user to select the method and date for checkout
*****
--%>

<!-- BEGIN OrderShippingMethodSelection.jsp -->
```

```

<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="http://commerce.ibm.com/base" prefix="wbase" %>
<%@ taglib uri="http://commerce.ibm.com/foundation" prefix="wcf" %>

<%@ include file="../../include/parameters.jspf" %>
<%@ include file="../../include/JSTLEnvironmentSetup.jspf" %>
<%@ include file="../../include/ErrorMessageSetup.jspf" %>

<!-- Required variables for breadcrumb support -->
<c:set var="shoppingcartPageGroup" value="true" scope="request"/>
<c:set var="shippingMethodSelectionPage" value="true" scope="request"/>
<!-- Standard parameters -->
<c:set var="storeId" value="${WParam.storeId}" />
<c:set var="catalogId" value="${WParam.catalogId}" />
<c:set var="addressId" value="${WParam.addressId}" />

<!-- Retrieve Data -->

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML Basic 1.1//EN"
    "http://www.w3.org/TR/xhtml1-basic/xhtml1-basic11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
    lang="${shortLocale}" xml:lang="${shortLocale}">
<head>
    <title>
        <fmt:message key="SHIPPING_METHOD_SELECTION_TITLE" bundle="${storeText}"/>
        - <c:out value="${storeName}"/></title>
    </title>
    <meta http-equiv="content-type" content="application/xhtml+xml" />
    <meta http-equiv="cache-control" content="max-age=300" />
    <meta name="viewport"
        content="width=device-width, initial-scale=1.0, user-scalable=no" />
    <link rel="stylesheet" type="text/css" href="${cssPath}" />
</head>

<body>
    <div id="wrapper">
        <%@ include file="../../include/HeaderDisplay.jspf" %>
        <%@ include file="../../include/BreadCrumbTrailDisplay.jspf" %>
        <div id="content_box" class="content_box">
<!-- content goes here -->
            </div><!-- #content_box .content_box -->
            <%@ include file="../../include/FooterDisplay.jspf" %>
        </div><!-- #wrapper -->
    </div>

```

```
</body>
</html>
<!-- END OrderShippingMethodSelection.jsp -->
```

This example code defines the JSP tag libraries that you will use, sets up the header with the title, and creates an empty body, including the page header and footer.

You first need to retrieve the data that is needed to display the page. Apart from generic date information, you need to generate a list of valid shipping modes using the IBM_UsableShippingInfo access profile for the findCurrentShoppingCart service of the Order component service module.

Furthermore, in order to pre-fill the selections with the order's current shipping information, if any, you need to retrieve order details from the database. In our scenario, we use the same findCurrentShoppingCart expression but with the IBM_Details access profile, as shown in Example 5-31.

Example 5-31 Service invocation of the findCurrentShoppingCart service to retrieve shipping information

```
<%-- 1. Get the shipping information for the current order --%>
<wcf:getData type="com.ibm.commerce.order.facade.datatypes.OrderType"
  var="shipDetails" expressionBuilder="findCurrentShoppingCart">
  <wcf:param name="accessProfile" value="IBM_UsableShippingInfo" />
</wcf:getData>
<wcf:getData type="com.ibm.commerce.order.facade.datatypes.OrderType"
  var="order" expressionBuilder="findCurrentShoppingCart">
  <wcf:param name="accessProfile" value="IBM_Details" />
</wcf:getData>
<%-- 1. End Get the shipping information for the current order --%>
```

7. To insert the service invocation code, in the JSP editor for the OrderShippingMethodSelection.jsp file, locate the following line comment:

```
<%-- Retrieve Data --%>
```
8. Insert the text in Example 5-31 just below the line comment. Then, press Ctrl+S to save the file, but do *not* close the file yet.
9. Now that the data that is needed to display the available shipping modes has been retrieved, you can display the data. To add the shipping method display code to the JSP, in the JSP editor for the OrderShippingMethodSelection.jsp file, locate the following line comment:

```
<!-- content goes here -->
```

10. Add the code shown in Example 5-32 after this line. Then, press Ctrl+S to save the file, but do *not* close the file yet.

Example 5-32 Shipping mode selection code for OrderShippingMethodSelection.jsp

```
<%-- 2. Begin Shipping Mode Selection --%>
<form id="shippingMethod"
      action="OrderChangeServiceShipInfoUpdate"
      onSubmit="return validate(this)">
  <input type="hidden" name="storeId" value="${storeId}" />
  <input type="hidden" name="catalogId" value="${catalogId}" />
  <input type="hidden" name="addressId" value="${addressId}" />
  <input type="hidden" name="requestedShipDate" value="" />
  <input type="hidden" name="ShipAsComplete" value="Y" />
  <c:forEach var="orderItem" items="${shipDetails.orderItem}" varStatus="status">
    <input type="hidden"
          name="orderItemId_${status.count}"
          value="${orderItem.orderItemIdentifier.uniqueID}" />
    <input type="hidden" id="shipModeId_${status.count}"
          name="shipModeId_${status.count}" value="" />
  </c:forEach>
  <input type="hidden" name="URL" value="mOrderBillingAddressSelection" />
  <input type="hidden" name="errorViewName" value="mOrderShippingMethodSelection" />
<div class="heading_container_with_underline">
  <h2>
    <fmt:message key="SHIPPING_METHOD" bundle="${storeText}" />
  </h2>
  <div class="clear_float"></div>
  <c:if test="${! empty errorMessage}">
    <div id="serverError" class="error"><c:out value="${errorMessage}" /></div>
  </c:if>
</div>
<p class="paragraph_blurb">
  <fmt:message key="SHIPPING_METHOD_SELECT" bundle="${storeText}" />
  <div id="shipMethodError" class="error"></div>
</p>
<ul class="entry">
  <%-- determine the preselected shipping mode --%>
  <c:set var="currentShippingInfo"
        value="${order.orderItem[0].orderItemShippingInfo}" />
  <c:choose>
    <%-- request parameters win (in case of a server error message) --%>
    <c:when test="${!empty WCPParam.shipModeId}">
      <c:set var="currentShippingModeId" value="${WCPParam.shipModeId}" />
    </c:when>
```

```

<%-- if no request parameter, get the mode from the order info --%>
<c:otherwise>
  <c:set var="currentShippingModeId"
value="\${currentShippingInfo.shippingMode.shippingModeIdentifier.uniqueID}" />
</c:otherwise>
</c:choose>
<c:forEach var="shippingMode"
  items="\${shipDetails.orderItem[0].usableShippingMode}">
  <c:set var="shipModeId" value="\${shippingMode.shippingModeIdentifier}" />
  <c:set var="shipModeExtId" value="\${shipModeId.externalIdentifier}"/>
  <c:set var="uniqueID" value="\${shipModeId.uniqueID}"/>

  <%-- preselect the current mode --%>
  <c:choose>
    <c:when test="\${(uniqueID eq currentShippingModeId)}">
      <c:set var="optionChecked" value="checked=&quot;checked&quot;" />
    </c:when>
    <c:otherwise>
      <c:set var="optionChecked" value="" />
    </c:otherwise>
  </c:choose>

  <%-- Show all the shipping options available except for pickUp in Store --%>
  <c:if test="\${shipModeExtId.shipModeCode != 'PickupInStore'}">
    <c:set var="radioBtnId" value="shipModeBtn_\${uniqueID}" />
    <li>
      <div class="radio_container">
        <input type="radio" id="\${radioBtnId}" name="shipModeId"
          value="\${uniqueID}" \${optionChecked} />
        <label for="\${radioBtnId}">
          <c:out value="\${shippingMode.description.value}"/>
        </label>
      </div>
    </li>
  </c:if>
</c:forEach>
</ul>
<%-- 2. End Shipping Mode Selection --%>

```

In addition to setting up headers and defining the beginning of the form for submitting the shipping method, the code in this example also displays any error message that is received from the server in the special error <div> with the ID of serverError. Note that this is separate from the empty shipMethodError <div> a few lines further down because we show form

validation errors caught by JavaScript in the context where they are found, while the server error messages do not carry context information.

As shown in Figure 5-23 on page 225, in addition to the shipping method selection, the `OrderShippingMethodSelection.jsp` page also contains a check box that allows users to request the addition of shipping instructions, as well as the ability to delay shipping to a future date.

11. To add the shipping instructions check box, in the JSP editor for the `OrderShippingMethodSelection.jsp` file, locate the following line comment:

```
<%-- 2. End Shipping Mode Selection --%>
```

12. Insert the text in Example 5-33 just below the line comment. Then, press Ctrl+S to save the file, but do *not* close the file yet.

Example 5-33 Shipping instructions check box for `OrderShippingMethodSelection.jsp`

```
<%-- 3. Begin Shipping Instructions --%>
<div class="heading_container_with_underline">
  <h2>
    <fmt:message key="SHIPPING_INSTRUCTIONS_OPT" bundle="${storeText}" />
  </h2>
  <div class="clear_float"></div>
</div>

<div id="shippingInstructions" class="entry">

  <%-- determine if the box should be checked by default --%>
  <c:choose>
    <%-- checked in the request params (was checked by the page was redisplayed
        due to a server-side error --%>
    <c:when test="${!empty WCPParam.addShippingInstr}">
      <input type="checkbox" id="addShippingInstr"
        name="addShippingInstr" checked="checked">
    </c:when>
    <%-- shipping instructions already specified on the order, so precheck --%>
    <c:when test="${!empty currentShippingInfo.shippingInstruction}">
      <input type="checkbox" id="addShippingInstr"
        name="addShippingInstr" checked="checked">
    </c:when>
    <%-- don't precheck --%>
    <c:otherwise>
      <input type="checkbox" id="addShippingInstr" name="addShippingInstr">
    </c:otherwise>
  </c:choose>

  <label for="addShippingInstr">
```

```

        <fmt:message key="SHIPPING_INSTRUCTIONS_ADD" bundle="${storeText}" />
    </label>

</div>
<!-- 3. End Shipping Instructions -->

```

13. To add the future shipping date options, in the JSP editor for the `OrderShippingMethodSelection.jsp` file, locate the following line comment:

```
<!-- 3. End Shipping Instructions -->
```

14. Insert the text in Example 5-34 just below the line comment. Press Ctrl+S to save the file, but do *not* close the file yet.

Example 5-34 Future shipping date options for OrderShippingMethodSelection.jsp

```

<!-- 4. Begin Shipping Date -->
<div id="shippingDate" class="entry">

    <p class="paragraph_blurb">
        <fmt:message key="SHIPPING_DATE_SELECT" bundle="${storeText}" />
        <div id="shipDateError" class="error"></div>
    </p>

    <!-- Get current date -->
    <jsp:useBean id="now" class="java.util.Date" scope="page" />
    <!-- Extract the year -->
    <fmt:formatDate var="startYear" value="${now}" pattern="yyyy" />
    <!-- add two years as the max into the future -->
    <c:set var="endYear" value="${startYear+2}" />

    <!-- retrieve the current shipping date from the order -->
    <c:set var="currentShipDate" value="${currentShippingInfo.requestedShipDate}" />
    <!-- extract the year, month and day from the date (which is a string) -->
    <c:if test="${!empty currentShipDate}">
        <!-- pattern="yyyy-mm-ddTHH:MM:ss.SSSZ" -->
        <c:set var="currentYear" value="${fn:substring(currentShipDate,0,4)}" />
        <c:set var="currentMonth" value="${fn:substring(currentShipDate,5,7)}" />
        <!-- remove prefixed zero -->
        <c:if test="${fn:substring(currentMonth,0,1) eq '0'}">
            <c:set var="currentMonth" value="${fn:substring(currentMonth,1,2)}" />
        </c:if>
        <c:set var="currentDay" value="${fn:substring(currentShipDate,8,10)}" />
        <!-- remove prefixed zero -->
        <c:if test="${fn:substring(currentDay,0,1) eq '0'}">
            <c:set var="currentDay" value="${fn:substring(currentDay,1,2)}" />
        </c:if>
    </c:if>

```

```

</c:if>
<%-- override with whatever selection was already done --%>
<c:if test="${!empty WParam.shipDay}">
    <c:set var="currentDay" value="${WParam.shipDay}" />
</c:if>
<c:if test="${!empty WParam.shipMonth}">
    <c:set var="currentMonth" value="${WParam.shipMonth}" />
</c:if>
<c:if test="${!empty WParam.shipYear}">
    <c:set var="currentYear" value="${WParam.shipYear}" />
</c:if>

<select id="shipDay" name="shipDay">
    <option value="">
        <fmt:message key="DAY_SELECT" bundle="${storeText}" />
    </option>
    <c:forEach var="day" begin="1" end="31">
        <c:choose>
            <c:when test="${day eq currentDay}">
                <option value="${day}" selected="selected">${day}</option>
            </c:when>
            <c:otherwise>
                <option value="${day}">${day}</option>
            </c:otherwise>
        </c:choose>
    </c:forEach>
</select>

<select id="shipMonth" name="shipMonth">
    <option value="">
        <fmt:message key="MONTH_SELECT" bundle="${storeText}" />
    </option>
    <c:forEach var="month" begin="1" end="12">
        <c:choose>
            <c:when test="${month eq currentMonth}">
                <option value="${month}" selected="selected">
            </c:when>
            <c:otherwise>
                <option value="${month}">
            </c:otherwise>
        </c:choose>
        <fmt:message key="MONTH_${month}" bundle="${storeText}" />
    </option>
    </c:forEach>
</select>

```



```

<select id="shipYear" name="shipYear">
  <option value="">
    <fmt:message key="YEAR_SELECT" bundle="${storeText}" />
  </option>
  <c:forEach var="year" begin="${startYear}" end="${endYear}">
    <c:choose>
      <c:when test="${year eq currentYear}">
        <option value="${year}" selected="selected">${year}</option>
      </c:when>
      <c:otherwise>
        <option value="${year}">${year}</option>
      </c:otherwise>
    </c:choose>
  </c:forEach>
</select>

</div>

<input type="submit" id="continuecheckout" name="continuecheckout"
  value="<fmt:message key="CONTINUE_CHECKOUT" bundle="${storeText}" />"
  class="input_button_float" />

</form>
<!-- 4. End Shipping Date -->

```

15. The only missing code from the JSP now is validation of the input to stop the form from being submitted if it is obviously wrong. To add this validation, in the JSP editor for the `OrderShippingMethodSelection.jsp` file, locate the following line, which specifies the end of the HTML header:

```
</head>
```

16. Insert the text in Example 5-35 just *before* the line above. Press Ctrl+S followed by Ctrl+W to save and close the file.

Example 5-35 JavaScript code for validating the shipping method form

```

<!-- 5. Begin Form validation and submit -->
<script type="text/javascript">
  <![CDATA[
  // reset all calculated form fields and error messages
  function resetForm(form) {
    document.getElementById('shipMethodError').innerHTML = '';
    document.getElementById('shipDateError').innerHTML = '';

    form.requestedShipDate.value = '';
  }
  ]>

```

```

}

function formatDate(date) {
    var formattedDate = date.getFullYear()+'-';
    var month = date.getMonth()+1;
    var day = date.getDate();
    if (month < 10) {
        formattedDate += '0';
    }
    formattedDate += month+'-';

    if (day < 10) {
        formattedDate += '0';
    }
    formattedDate += day;

    return formattedDate;
}

// validate the form and display error messages
// if there are problems with the input
function validate(form) {

    resetForm(form);

    // status variable, specifying whether the form is ready to be submitted
    var submitform = true;

    // determine if a shipping method has been selected
    var selectedShipModeId = '';
    for (var i = form.shipModeId.length-1; i >= 0; --i) {
        if (form.shipModeId[i].checked) {
            selectedShipModeId = form.shipModeId[i].value;
            break;
        }
    }

    // if not, display error and stay on the page
    if (selectedShipModeId == '') {
        document.getElementById('shipMethodError').innerHTML =
            "<fmt:message key='SHIPPING_METHOD_MISSING' bundle='${storeText}'/>";
        submitform = false;
    }
    else {
        // A method has been selected. Now set this value on all the shipModeId_n

```

```

    // hidden fields. We need to do this as the order change service wants
    // a shipmode for each and every order line
    for (var i = 1; i <= ${fn:length(shipDetails.orderItem)}; i++) {
        document.getElementById('shipModeId_'+i).value = selectedShipModeId;
    }
}

// if one of the date drop-downs have been selected, all must be
var day = form.shipDay.value;
var month = form.shipMonth.value;
var year = form.shipYear.value;
var now = new Date();
if (day != "" || month != "" || year != "") {
    if (day == "" || month == "" || year == "") {
        document.getElementById('shipDateError').innerHTML =
            "<fmt:message key='SHIPPING_DATE_MISSING_FIELDS'
                bundle='${storeText}'/>";
        submitform = false;
    }
    else {

        // validate the date format, utilizing a JavaScript ECMA standard
        // quirk in that using the "mm/dd/yyyy" constructor, JavaScript will
        // automatically roll to the next correct date if specifying a wrong
        // date...
        var date = new Date(month + "/" + day + "/" + year);

        if (date.getMonth()+1 != month ||
            date.getDate() != day ||
            date.getFullYear() != year) {

            // if the date somehow changed, the initial values were not valid.
            document.getElementById('shipDateError').innerHTML =
                "<fmt:message key='SHIPPING_DATE_WRONG' bundle='${storeText}'/>";
            submitform = false;
        }
        else if (date <= now) {
            // display error if shipping is requested today or earlier
            document.getElementById('shipDateError').innerHTML =
                "<fmt:message key='SHIPPING_DATE_EARLY' bundle='${storeText}'/>";
            submitform = false;
        }
    }

    // date is fine. Put it into the hidden field in the format expected by
    // the shipping info update service...

```

```

        form.requestedShipDate.value = formatDate(date);
    }
}
else {
    // all fields are empty. Prefill with tomorrow's date
    now.setDate(now.getDate()+1);
    form.requestedShipDate.value = formatDate(now);
}

// redirect to the shipping instructions details if the check box is checked
if (form.addShippingInstr.checked) {
    form.URL.value = 'mOrderShippingInstructions';
}
else {
    form.URL.value = 'mOrderBillingAddressSelection';
}














return submitform;
}
//]]>
</script>
<!-- 5. End Form validation and submit -->

```

The shipping method selection JSP is now complete, except for the resource bundles for the page that you will add, as described in “Add localized texts” on page 284.

Create the OrderShippingInstructions.jsp file

To create the new OrderShippingMethodSelection.jsp file:

1. Open the development environment by selecting **Start**  **Programs**  **IBM**  **WebSphere**  **WebSphere Commerce**  **WebSphere Commerce Developer Enterprise**.
2. Switch to the Java EE perspective if it is not already active:
 - a. Select **Window**  **Open Perspective**  **Other**.
 - b. The Open Perspective window opens. Select **Java EE**, and click **OK**.
3. In the Enterprise Explorer, expand and right-click **Stores**  **WebContent**  **Madisons**  **mobile**  **ShoppingArea**  **CheckoutSection**.
4. In the context menu that opens, select **New**  **File**.

5. The New File dialog box opens. Enter `OrderShippingInstructions.jsp` in the File name field, as shown in Figure 5-35, and click **Finish**.

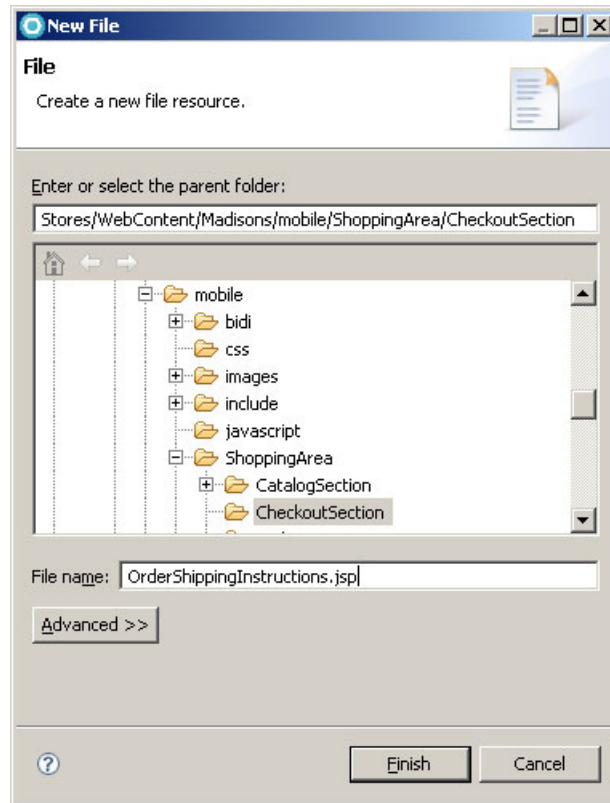


Figure 5-35 Creating the `OrderShippingInstructions.jsp` file using the New File wizard

6. The JSP file editor opens in the source tab and shows a blank page. Enter the skeleton JSP file shown in Example 5-36 into the page, and press `Ctrl+S` to save the file. Do *not* close the file yet.

Example 5-36 Skeleton for the `OrderShippingInstructions.jsp` file

```
<%--
    *****
    * This JSP displays a text area, allowing the shopper to
    * enter specific shipping instructions
    *****
--%>

<!-- BEGIN OrderShippingInstructions.jsp -->
```

```

<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="http://commerce.ibm.com/base" prefix="wbase" %>
<%@ taglib uri="http://commerce.ibm.com/foundation" prefix="wcf" %>

<%@ include file="../../include/parameters.jspf" %>
<%@ include file="../../include/JSTLEnvironmentSetup.jspf" %>
<%@ include file="../../include/ErrorMessageSetup.jspf" %>

<!-- Required variables for breadcrumb support -->
<c:set var="shoppingcartPageGroup" value="true" scope="request"/>
<c:set var="shippingInstructionsPage" value="true" scope="request"/>
<!-- Standard parameters -->
<c:set var="storeId" value="${WCPParam.storeId}" />
<c:set var="catalogId" value="${WCPParam.catalogId}" />
<!-- Retrieve Data -->
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML Basic 1.1//EN"
    "http://www.w3.org/TR/xhtml1-basic/xhtml1-basic11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
    lang="${shortLocale}" xml:lang="${shortLocale}">
<head>
    <title>
        <fmt:message key="SHIPPING_INSTRUCTIONS_TITLE" bundle="${storeText}"/>
        - <c:out value="${storeName}"/>
    </title>
    <meta http-equiv="content-type" content="application/xhtml+xml" />
    <meta http-equiv="cache-control" content="max-age=300" />
    <meta name="viewport"
        content="width=device-width, initial-scale=1.0, user-scalable=no" />
    <link rel="stylesheet" type="text/css" href="${cssPath}" />
</head>
<body>
    <div id="wrapper">
        <%@ include file="../../include/HeaderDisplay.jspf" %>
        <%@ include file="../../include/BreadCrumbTrailDisplay.jspf" %>
        <div id="content_box" class="content_box">
<!-- content goes here -->
        </div><!-- #content_box .content_box -->
        <%@ include file="../../include/FooterDisplay.jspf" %>
        </div><!-- #wrapper -->
    </body>
</html>
<!-- END OrderShippingInstructions.jsp -->

```

The code in this example defines the JSP tag libraries that you use, sets up the header with title, and creates an empty body that includes the page header and footer.

As shown in Figure 5-23 on page 225, the shipping instruction JSP is very simple and contains only a text area and a submit button. In addition to the visible areas, we also add JavaScript validation in this scenario, as we did for the `OrderShippingMethodSelection.jsp` shipping method. We do, however, still need to retrieve data for the page. Specifically, we need to retrieve order information in order to determine the identifiers for the order lines, to retrieve the current shipping instructions, if any, and to initialize the maximum number of characters that allowed for the shipping instructions.

7. In the JSP editor for the `OrderShippingInstructions.jsp` file, locate the following line:

```
<%-- Retrieve Data --%>
```
8. Add the code shown in Example 5-37 after this line. Press Ctrl+S to save the file, but do *not* close the file yet.

Example 5-37 Code to retrieve shipping information for the current order

```
<%-- 1. Begin Get the shipping information for the current order --%>
<wcf:getData type="com.ibm.commerce.order.facade.datatypes.OrderType"
  var="order" expressionBuilder="findCurrentShoppingCart">
  <wcf:param name="accessProfile" value="IBM_Details" />
</wcf:getData>
<c:set var="currentInstructions"
  value="{order.orderItem[0].orderItemShippingInfo.shippingInstruction}" />
<c:if test="{!empty WParam.shipInstructions}">
  <c:set var="currentInstructions" value="{WParam.shipInstructions}" />
</c:if>
<c:set var="shipInstructionsMaxLength" value="200" />
<%-- 1. End Get the shipping information for the current order --%>
```

The code in this example retrieves the shipping details for the current order and sets up a variable with the current shipping instructions for the first order item. Furthermore, the code defines the maximum length that will be accepted for the shipping instructions. We set an arbitrary length of 200 characters here, although the database field that holds the instructions can accept up to 4,000 characters. If you want, you can increase this limit to suit your needs.

The discussion in 5.4.4, “Design new shopping flow” on page 239 revealed that we need to use the `OrderChangeServiceShipInfoUpdate` struts action to update the shipping instructions, as we did when we update the `OrderShippingMethodSelection.jsp` shipping method and date.

9. In the JSP editor for the `OrderShippingInstructions.jsp` file, locate the following line:

`<!-- content goes here -->`

10. Add the code shown in Example 5-38 after this line. Press Ctrl+S to save the file, but do *not* close the file yet.

Example 5-38 Shipping instruction code for `OrderShippingInstructions.jsp`

```
<%-- 2. Begin Shipping Instructions --%>
<form id="shippingMethod"
      action="OrderChangeServiceShipInfoUpdate"
      onSubmit="return validate(this)">
  <input type="hidden" name="storeId" value="${storeId}" />
  <input type="hidden" name="catalogId" value="${catalogId}" />
  <input type="hidden" name="ShipAsComplete" value="Y" />
  <input type="hidden"
        name="orderId"
        value="${shipDetails.orderIdentifier.uniqueID}" />
  <c:forEach var="orderItem" items="${shipDetails.orderItem}" varStatus="status">
    <input type="hidden"
          name="orderItemId_${status.count}"
          value="${orderItem.orderItemIdentifier.uniqueID}" />
    <input type="hidden" id="shipInstructions_${status.count}"
          name="shipInstructions_${status.count}" value="" />
  </c:forEach>

  <input type="hidden" name="URL" value="mOrderBillingAddressSelection" />
  <input type="hidden" name="errorViewName" value="mOrderShippingInstructions" />

  <div class="heading_container_with_underline">
    <h2>
      <fmt:message key="SHIPPING_INSTRUCTIONS" bundle="${storeText}" />
    </h2>
    <div class="clear_float"></div>
  </div>

  <p class="paragraph_blurb">
    <fmt:message key="SHIPPING_INSTRUCTIONS_ENTER" bundle="${storeText}" />
    <div id="shipInstructionsError" class="error">
      <c:out value="${errorMessage}" />
    </div>
  </p>
  <textarea id="shipInstructions" name="shipInstructions"
            rows="5" cols="28"><c:out value="${currentInstructions}" /></textarea>
  <p class="paragraph_blurb">
```



```

        <fmt:message key="SHIPPING_INSTRUCTIONS_HELP" bundle="${storeText}"/>
    </p>

    <input type="submit" id="continuecheckout" name="continuecheckout"
        value="<fmt:message key="CONTINUE_CHECKOUT" bundle="${storeText}" />"
        class="input_button_float" />

</form>
<!-- 2. End Shipping Instructions -->

```

The code in this example displays the server error message within the `shipInstructionsError` `<div>` that is also used for JavaScript validation. This process is in contrast with the error message setup in the shipping method selection JSP that we set up earlier. We can make this change here, because there is only one form field and, thus, the context of the error is defined implicitly.

In addition to the shipping instruction entry form, we also add JavaScript validation. The validation is limited to checking whether there has been any entry and whether the entry exceeds the maximum allowed number of characters.

11. In the JSP editor for the `OrderShippingInstructions.jsp` file, locate the following line, which specifies the end of the HTML header:

```
</head>
```

12. Insert the text in Example 5-39 just *before* this line. Press Ctrl+S followed by Ctrl+W to save and close the file.

Example 5-39 JavaScript code to validate the shipping instructions form

```

<!-- 3. Begin JavaScript validation and submission code -->
<script type="text/javascript">
//
// reset all error messages
function resetErrors() {
    document.getElementById('shipInstructionsError').innerHTML = '';
}

// validate the form and display error messages
// if there are problems with the input
function validate(form) {

    resetErrors();

    // status variable, specifying whether the form is ready to be submitted
    var submitform = true;
</pre>
</div>
<div data-bbox="350 942 852 961" data-label="Page-Footer">
<p>Chapter 5. Mobile commerce features in WebSphere Commerce V7 <b>283</b></p>
</div>
```

```

var shipInstructions = form.shipInstructions.value;










if (!shipInstructions || shipInstructions.length == 0) {
    document.getElementById('shipInstructionsError').innerHTML =
        "<fmt:message key='SHIPPING_INSTRUCTIONS_EMPTY' bundle='${storeText}' />";
    submitform = false;
}
else if (shipInstructions && shipInstructions.length >
    ${shipInstructionsMaxLength}) {
    document.getElementById('shipInstructionsError').innerHTML =
        "<fmt:message key='SHIPPING_INSTRUCTIONS_TOO_LONG' bundle='${storeText}'
/>";
    submitform = false;
}
else {
    // set the instructions for each and every order line
    for (var i = 1; i <= ${fn:length(shipDetails.orderItem)}; i++) {
        document.getElementById('shipInstructions_'+i).value = shipInstructions;
    }
}
return submitform;
}
//]]>
</script>
<!-- 3. End JavaScript validation and submission code -->

```

The shipping instructions JSP is now complete.

Add localized texts

The JSPs created in the previous sections made references to various text snippets for display in the storefront. You need to add these texts to the properties file for the mobile store. In this example, we create only the English texts. To add the default English texts to the Madisons Mobile Starter Store texts:

1. Open the development environment by selecting **Start**  **IBM**  **WebSphere**  **WebSphere Commerce**  **Developer Enterprise**.
2. Switch to the Java EE perspective if it is not already active:
 - a. Select **Window**  **Open Perspective**  **Other**.
 - b. The Open Perspective window opens. Select **Java EE**, and click **OK**.
3. In the Enterprise Explorer, expand and double-click **Stores**  **Java Resources: src**  **Madisons.mobile**  **storetext.properties**.

Important: Ensure that you double-click the `storetext.properties` file in the `Madisons.mobile` folder and *not* the `Madisons` folder.

4. When the file opens, insert the properties from Example 5-40 at the end of the file. Then, press Ctrl+S followed by Ctrl+W to save and close the file.

Example 5-40 NLS texts

```
#-----  
# Shop online - BEGIN  
#-----  
# Order Shipping Address Selection  
SHIPPING_ADDRESS_SELECT_TITLE = Shipping Address  
YOUR_SHIPPING_ADDRESSES = Your Shipping Addresses  
SHIPPING_ADDRESS_SELECT = Select the shipping address for your order.  
SHIPPING_ADDRESS_CREATE = To use a new shipping address, click "Create new address".  
# Order Shipping Address Details  
SHIPPING_ADDRESS_DETAILS_TITLE = Shipping Address Details  
SHIPPING_ADDRESS_TITLE = Shipping Address  
# Order Shipping Method Selection  
SHIPPING_METHOD_SELECTION_TITLE = Shipping Method  
SHIPPING_METHOD = Shipping Method  
SHIPPING_METHOD_SELECT = Choose a shipping method  
SHIPPING_INSTRUCTIONS_OPT = Optional Shipping Instructions  
SHIPPING_INSTRUCTIONS_ADD = Select this check box to add special shipping  
instructions.  
SHIPPING_DATE_SELECT = To ship at a later time, choose a date below  
DAY_SELECT = Day  
MONTH_SELECT = Month  
YEAR_SELECT = Year  
MONTH_1 = Jan  
MONTH_2 = Feb  
MONTH_3 = Mar  
MONTH_4 = Apr  
MONTH_5 = May  
MONTH_6 = Jun  
MONTH_7 = Jul  
MONTH_8 = Aug  
MONTH_9 = Sep  
MONTH_10 = Oct  
MONTH_11 = Nov  
MONTH_12 = Dec  
# JavaScript error messages. These will be put inline in a literal string  
# using double-quotes, so escape any quote or newline characters as per
```

```

# JavaScript code formatting rules
SHIPPING_METHOD_MISSING = Please select a shipping method.
SHIPPING_DATE_MISSING_FIELDS = Select a full shipping date, or reset all date \
    fields to request shipment as soon as possible.
SHIPPING_DATE_WRONG = The entered date is not valid. Please correct the date.
SHIPPING_DATE_EARLY = The entered date is too early. Please select a later date.
# Order Shipping Instructions
SHIPPING_INSTRUCTIONS_TITLE = Shipping Instructions
SHIPPING_INSTRUCTIONS = Shipping Instructions
SHIPPING_INSTRUCTIONS_ENTER = Enter your shipping instructions below.
SHIPPING_INSTRUCTIONS_HELP = Max 135 characters. Examples: Leave on porch, \
    Leave with neighbor if no answer.
SHIPPING_INSTRUCTIONS_EMPTY = Please enter shipping instructions or go back to \
    skip this step.
SHIPPING_INSTRUCTIONS_TOO_LONG = The specified text for shipping instructions is \
    too long. Please limit to 200 characters.
#-----
# Shop online - BEGIN
#-----

```

Test the new pages

Now that you have created the views, set up access control, created the JSPs and, added the localized texts for the new pages, you can test the new pages. In our scenario, we test the pages through the Madisons Mobile Starter Store checkout flow and switch to the new custom shipping address selection page when the Store Locator opens.

To test the pages:

1. Start or restart the test server, as described in “Analyze the current checkout pages” on page 229.
2. Open a Web browser and navigate to a product page. We used Mozilla Firefox to navigate to a product in the catalog. We chose the violet Enzi Espresso Machine at the following URL:

```
http://localhost/webapp/wcs/stores/servlet/mProduct1_10051_10051_-1_10074_10345_10053_10053_catNav__
```

Note: The above link worked in our environment with the standard starter stores deployed at the time of installing IBM WebSphere Commerce V7 Developer Edition. If you published the Madisons Starter Store manually or changed the catalog, this link might not work for you. If this is the case, navigate to find a product that is in stock both online and in one of the brick-and-mortar stores.

3. From the product page click **Add to Cart**.
4. The Shopping Cart page opens with the product shown in the cart. Click **Proceed to Checkout** to continue the checkout process.

Note: Because you have not yet customized the Shopping Cart page, you have to accept the option of picking up in a store. You will change to the shipping pages manually at a later step during the test.

5. The sign in or checkout page opens. Click **Continue Checkout** to check out as a guest shopper.
6. The Store Locator opens. Skip to the new shipping address selection page by entering the following URL in the browser's address bar:

`https://localhost/webapp/wcs/stores/servlet/mOrderShippingAddressSelection?langId=-1&catalogId=10051&storeId=10051`

Note: If your store or catalog IDs are different, you need to substitute your values in this link.

7. The shipping address details page opens. Enter a new billing address, and click **Continue Checkout**.

We entered the following information:

Nick Name:	ITS0
First Name:	IBM Corporation
Last Name:	International Technical Support Organization
Street Address 1:	Dept. HYTD Mail Station P099
Street Address 2:	2455 South Road
City:	Poughkeepsie
Country/Region:	United States
State/Province:	New York
Zip code/ Postal code:	12601-5400
Phone number:	1-800-IBM-HELP
E-mail:	redbooks@us.ibm.com

8. The shipping address selection page opens with the new address. Select the address, and click **Continue Checkout**.
9. The shipping method selection page opens. Select a shipping method, select the "Select this check box to add special shipping instructions" option, and click **Continue Checkout**.

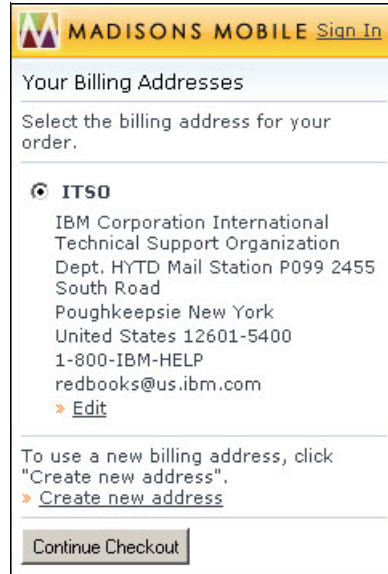
We selected the International Regular shipping method.

10. The shipping instructions page opens. Enter an instruction, and click **Continue Checkout**.

We entered the following text:

Please leave on porch if there is no answer.

11. Because the shipping address is also a valid billing address, the billing address selection page opens with the shipping address that you entered in step 7, as shown in Figure 5-36. Select the address as the billing address, and click **Continue Checkout**.



MADISONS MOBILE [Sign In](#)

Your Billing Addresses

Select the billing address for your order.

☒ **ITSO**

IBM Corporation International
Technical Support Organization
Dept. HYTD Mail Station P099 2455
South Road
Poughkeepsie New York
United States 12601-5400
1-800-IBM-HELP
redbooks@us.ibm.com
[» Edit](#)

To use a new billing address, click "Create new address".
[» Create new address](#)

[Continue Checkout](#)

Figure 5-36 Billing address selection page pre-filled with the shipping address

12. The Payment selection page opens. Note that the “Pay in Store” payment option is available. You will customize this page in 5.4.6, “Modify existing pages” on page 291 so that this option appears only when picking up from a store.

In the Payment dialog box shown in Figure 5-37:

- Select **Credit Card** as the Payment Method.
- Select **VISA Credit Card** as the Card type.
- Enter 4111111111111111 for the Card number.
- Select **01** for the Month and **2010** for the Year.
- Enter 123 for the CVV2 Number.
- Click **Continue Checkout**.

MADISONS MOBILE [Sign In](#)

Payment
Promotion Code
Enter a promotion code to receive further discounts.
Promotion Code:

Order Subtotal: \$179.99
Discount: (\$5.00)
Order Total: \$174.99

Payment Method
☐ Pay in Store
☒ Credit Card

Card type
VISA Credit Card

Card number
4111111111111111

Month **Year**
01 2010

CCV2 number
123

Figure 5-37 The payment method page with a test credit card number

13. The Order Summary page opens as shown in Figure 5-38. Notice the wrong label and edit link for the shipping address. We address this situation in 5.4.6, “Modify existing pages” on page 291. Click **Place Your Order**.


<div> MADISON'S MOBILE Sign In</div> <div>Home</div> <div>Order Summary Review your order and select the "Place Your Order" button at the bottom of the page.</div> <div>Store Location Your items can be picked up at the following location.</div> <div>ITSO IBM Corporation International Technical Support Organization Dept. HYTD Mail Station P099 2455 South Road Poughkeepsie New York United States 12601-5400 1-800-IBM-HELP redbooks@us.ibm.com</div> <div>Change Store</div>	<div>Change Store</div> <div>Billing Address ITSO IBM Corporation International Technical Support Organization Dept. HYTD Mail Station P099 2455 South Road Poughkeepsie New York United States 12601-5400 1-800-IBM-HELP redbooks@us.ibm.com</div> <div>Edit</div> <div>Billing Method Payment: VISA Credit Card Month: 01 Account number: *****11111 CVV2 number: ---</div>	<div>2010 Amount: \$174.99 Edit</div> <div>Items 1. Enzi Espresso Machine, Violet SKU: EI-0101V In-Stock Price: \$179.99 Qty: 1 Edit</div> <div>Order Subtotal: \$179.99 Discount: (\$5.00) Tax: \$0.00 Shipping: \$0.00 Order Total: \$174.99</div> <div>Place Your Order</div>
--	---	---

Figure 5-38 The order summary page with the wrong label and link for the shipping address

The order confirmation page opens as shown in Figure 5-39. Notice again the wrong label for the shipping address, which we will address in 5.4.6, “Modify existing pages” on page 291.

<div>  MADISONS MOBILE Sign In </div> <div> Home </div> <div> <p>Thank you for your order!</p> <p>You will receive a confirmation by e-mail to verify your order.</p> <p>Order number: 13501</p> <p>Order date: August 13, 2009</p> </div> <div> <p>Store Location</p> <p>Your items can be picked up at the following location.</p> <p>ITSO</p> <p>IBM Corporation International Technical Support Organization</p> <p>Dept. HYTD Mail Station P099 2455</p> <p>South Road</p> <p>Poughkeepsie New York</p> </div>	<p>United States 12601-5400</p> <p>1-800-IBM-HELP</p> <p>redbooks@us.ibm.com</p> <hr/> <p>Billing Address</p> <p>ITSO</p> <p>IBM Corporation International Technical Support Organization</p> <p>Dept. HYTD Mail Station P099 2455</p> <p>South Road</p> <p>Poughkeepsie New York</p> <p>United States 12601-5400</p> <p>1-800-IBM-HELP</p> <p>redbooks@us.ibm.com</p> <hr/> <p>Billing Method</p> <p>Payment: VISA Credit Card</p> <p>Month: 01</p> <p>Account number: *****11111</p> <p>CVV2 number: ---</p> <p>Year: 2010</p>	<p>Amount: \$174.99</p> <hr/> <p>Items</p> <p>1. Enzi Espresso Machine, Violet</p> <p>SKU: EI-0101V</p> <p>In-Stock</p> <p>Price: \$179.99</p> <p>Qty: 1</p> <hr/> <p>Order Subtotal: \$179.99</p> <p>Discount: (\$5.00)</p> <p>Tax: \$0.00</p> <p>Shipping: \$0.00</p> <p>Order Total: \$174.99</p> <hr/> <p>Continue Shopping</p>
--	--	--

Figure 5-39 Order confirmation page with the wrong label for the shipping address

For a production system, you obviously need more thorough testing, for example testing error conditions, using the browser’s back and forward buttons, and so forth. We did not perform extensive testing for this scenario. An example for further testing is to verify that the shipping instruction page shows only when the corresponding check box is checked on the shipping method page.

5.4.6 Modify existing pages

As the analysis, design, and testing has shown, you need to modify some of the existing pages to support the selection of online shopping in the Madisons Mobile Starter Store. In this section, we explain how to make the following modifications to the standard Madisons Mobile Starter Store checkout pages:

- ▶ Add checkout with shipping option to shopping cart
- ▶ Update the sign in or check out page
- ▶ Remove the option to pay in store from the payment page
- ▶ Modify the order summary page to show shipping address
- ▶ Modify order confirmation page to show shipping address














- ▶ Add breadcrumb support for new pages and flow
- ▶ Define localized strings for customized pages

Tip: In the following sections, we modify JSP pages that are part of the IBM WebSphere Commerce V7 Developer Edition standard installation. Thus, we recommend that you back up these files before you make any modifications to them so that you can always return to the original configuration.

Add checkout with shipping option to shopping cart

At this point, the shopping cart for the Madisons Mobile Starter Store still has only one option for checkout. In this section, we explain how to add an additional option for checkout with shipping, which results in the page as shown in Figure 5-24 on page 226.

To customize the Shopping Cart page:

1. Open the development environment by selecting **Start**  **Programs**  **IBM**  **WebSphere**  **WebSphere Commerce**  **WebSphere Commerce Developer Enterprise**.
2. Switch to the Java EE perspective if it is not already active:
 - a. Select **Window**  **Open Perspective**  **Other**.
 - b. The Open Perspective window opens. Select **Java EE**, and click **OK**.
3. In the Enterprise Explorer, expand and double-click **Stores**  **WebContent**  **Madisons**  **mobile**  **ShoppingArea**  **ShopcartSection**  **OrderItemDisplay.jsp**.
4. The file opens in the JSP source code editor. You need to locate the code that redirects to the sign in or checkout page. The view name for that page is `mCheckoutLogon`, as we discovered during the initial analysis. To locate this code, press Ctrl+F, enter `mCheckoutLogon` in the Find/Replace dialog box, and click **Find**.
5. The code snippet shown in Example 5-41 is displayed. (We reformatted the code for readability here.) In this example, we highlighted the search term that was found. Click **Close** in the Find/Replace window.

Example 5-41 Code for generating the URL for the next page in the checkout flow

```
<%-- Bypass checkout logon when already signed in --%>
<c:choose>
  <c:when test="${userType == 'G'}">
    <wcf:url var="CheckoutLogon" value="mCheckoutLogon">
      <wcf:param name="langId" value="${langId}" />
      <wcf:param name="storeId" value="${WCPParam.storeId}" />
      <wcf:param name="catalogId" value="${WCPParam.catalogId}" />
    </wcf:url>
```

```

</c:when>
<c:otherwise>
  <wcf:url var="CheckoutLogon" value="mSelectedStoreListView">
    <wcf:param name="langId" value="{langId}" />
    <wcf:param name="storeId" value="{WCPParam.storeId}" />
    <wcf:param name="catalogId" value="{WCPParam.catalogId}" />
    <wcf:param name="fromPage" value="ShoppingCart" />
  </wcf:url>
</c:otherwise>
</c:choose>

<div id="shipping_options">
  <div class="radio_container"><input type="radio" name="shipping_option"
id="pick_up_at_store" checked /> <label for="pick_up_at_store"><fmt:message
key="PICK_UP_AT_STORE" bundle="{storeText}" /></label></div>
  <button type="button"
onclick="window.location.href='{CheckoutLogon}'"><fmt:message
key="PROCEED_TO_CHECKOUT" bundle="{storeText}" /></button>
</div>

```

Example 5-41 shows that the variable `CheckoutLogon` is set to the URL for the sign in or checkout page if the current user is found to be a guest user. Otherwise, the variable is set to the URL of the store selection list.

The value of the `CheckoutLogon` variable is then used in the `onClick` event of the submit button.

Example 5-41 also shows that the sign in or checkout page itself must have logic that redirects to the store selection page, because the link to the sign in or checkout page is not given the URL for the store selection page. We explain how to modify this link in a later section.

We use two hidden HTML forms to submit the page, one for each checkout scenario. A simple JavaScript function verifies the radio button setting and submits the appropriate form.

We start by defining the revised HTML form for the current pick up in store action.

6. In the editor for the `OrderItemDisplay.jsp` file, replace the code shown in Example 5-41 with the code from Example 5-42. Then, press Ctrl+S to save the changes, but do *not* close the editor window.

Example 5-42 Code to define the HTML form for check out with pick up in a store

```
<wcf:url var="storeListURL" value="mSelectedStoreListView">
  <wcf:param name="langId" value="{langId}" />
  <wcf:param name="storeId" value="{WCPParam.storeId}" />
  <wcf:param name="catalogId" value="{WCPParam.catalogId}" />
  <wcf:param name="orderId" value="{order.orderIdentifier.uniqueID}" />
  <wcf:param name="fromPage" value="ShoppingCart" />
</wcf:url>
<wcf:url var="shippingURL" value="mOrderShippingAddressSelection">
  <wcf:param name="langId" value="{langId}" />
  <wcf:param name="storeId" value="{WCPParam.storeId}" />
  <wcf:param name="catalogId" value="{WCPParam.catalogId}" />
</wcf:url>

<div id="shipping_options">
  <div class="radio_container">
    <input type="radio" name="shipping_option" id="pick_up_at_store" checked />
    <label for="pick_up_at_store">
      <fmt:message key="PICK_UP_AT_STORE" bundle="{storeText}" />
    </label>
  </div>
  <div class="radio_container">
    <input type="radio" name="shipping_option" id="shop_online" />
    <label for="shop_online">
      <fmt:message key="SHOP_ONLINE" bundle="{storeText}" />
    </label>
  </div>
  <button type="button" onclick="doCheckout()">
    <fmt:message key="PROCEED_TO_CHECKOUT" bundle="{storeText}" />
  </button>

  <c:choose>
    <c:when test="{userType != 'G'}">
      <form id="pick_up_at_store_form" action="mSelectedStoreListView">
        <input type="hidden" name="fromPage" value="ShoppingCart" />
        <input type="hidden" name="langId" value="{langId}" />
        <input type="hidden" name="storeId" value="{WCPParam.storeId}" />
        <input type="hidden" name="catalogId" value="{WCPParam.catalogId}" />
      </form>
      <form id="shop_online_form" action="mOrderShippingAddressSelection">
```

```

        <input type="hidden" name="langId" value="{langId}" />
        <input type="hidden" name="storeId" value="{WCPParam.storeId}" />
        <input type="hidden" name="catalogId" value="{WCPParam.catalogId}" />
    </form>
</c:when>
<c:otherwise>
    <form id="pick_up_at_store_form" action="mCheckoutLogon">
        <input type="hidden" name="nextURL" value="{storeListURL}" />
        <input type="hidden" name="langId" value="{langId}" />
        <input type="hidden" name="storeId" value="{WCPParam.storeId}" />
        <input type="hidden" name="catalogId" value="{WCPParam.catalogId}" />
    </form>
    <form id="shop_online_form" action="mCheckoutLogon">
        <input type="hidden" name="nextURL" value="{shippingURL}" />
        <input type="hidden" name="langId" value="{langId}" />
        <input type="hidden" name="storeId" value="{WCPParam.storeId}" />
        <input type="hidden" name="catalogId" value="{WCPParam.catalogId}" />
    </form>
</c:otherwise>
</c:choose>
</div>

```

The code in this example defines a new radio button and two hidden forms with the information that is dependent on the shopper's user type.

7. Next, define the JavaScript function that submits the correct form. In the editor for the `OrderItemDisplay.jsp` file, insert the code shown in Example 5-43 between the `<head>` and `</head>` tags at the top of the file. Then, press Ctrl+S and Ctrl+W to save and close the file.

Example 5-43 Code to submit the correct form

```











<script type="text/javascript">
    //
    /**
     * Determines which checkout flow
     */
    function doCheckout() {
        if(document.getElementById("pick_up_at_store").checked) {
            document.getElementById("pick_up_at_store_form").submit();
        }
        else {
            document.getElementById("shop_online_form").submit();
        }
    }
    //]]&gt;
&lt;/script&gt;
</pre>
<hr/>
</div>
<div data-bbox="351 941 852 961" data-label="Page-Footer">
<p>Chapter 5. Mobile commerce features in WebSphere Commerce V7 <b>295</b></p>
</div>
```

The shopping cart JSP is now updated with the option for shopping online.

Update the sign in or check out page

As mentioned previously, the sign in or check out page contains logic to determine the next page in the checkout flow. Because must redirect to the new shipping address selection page if the shopper selects to shop online, we need to amend this page.

To analyze and amend the JSP:

1. Open the development environment by selecting **Start**  **Programs**  **IBM**  **WebSphere**  **WebSphere Commerce**  **WebSphere Commerce Developer Enterprise**.
2. Switch to the Java EE perspective if it is not already active:
 - a. Select **Window**  **Open Perspective**  **Other**.
 - b. The Open Perspective window opens. Select **Java EE**, and click **OK**.
3. In the Enterprise Explorer, expand and double-click **Stores**  **WebContent**  **Madisons**  **mobile** **UserArea** **AccountSection** **LogonSubsection** **CheckoutLogon.jsp**.
4. The file opens in the JSP source code editor. You need to locate the code that redirects to the store list. Press Ctrl+F, enter <button in the Find/Replace dialog box, and click **Find**.
5. The code snippet shown in Example 5-44 is displayed. The highlighted text in Example 5-44 shows that the target page is controlled by the nextURL variable. Click **Close** in the Find/Replace window.

Example 5-44 Code in the CheckoutLogon.jsp file for continuing the checkout flow

```
<form id="register_link" action="mStoreLocatorView">
  <fieldset>
    <p><fmt:message key="CHECKOUT_WITHOUT_SIGN_IN" bundle="${storeText}"/></p>
    <p><fmt:message key="GUEST_CHECKOUT_MESSAGE" bundle="${storeText}"/></p>
    <button type="button" onclick="window.location.href='${nextURL}'"><fmt:message
key="CONTINUE_CHECKOUT" bundle="${storeText}"/></button>
  </fieldset>
</form>
```

6. Press Ctrl+F, enter var="nextURL" in the Find/Replace window, and click **Find**.

7. The code snippet shown in Example 5-45 is displayed. Click **Close** in the Find/Replace window.

Example 5-45 Code to generate the URL for the next page in the checkout flow

```
<wcf:url var="nextURL" value="mSelectedStoreListView">
  <wcf:param name="langId" value="{langId}" />
  <wcf:param name="storeId" value="{WParam.storeId}" />
  <wcf:param name="catalogId" value="{WParam.catalogId}" />
  <wcf:param name="fromPage" value="ShoppingCart" />
</wcf:url>
```

8. Example 5-42 on page 294 introduced a nextURL parameter to the forms that redirects the user to this page from the Shopping Cart page. You need to overwrite the value of the nextURL variable if this parameter is specified. Insert the code snippet shown in Example 5-46 *after* the code shown in Example 5-45.

Example 5-46 Code to change the next page of the checkout flow dynamically

```
<c:if test="{!empty WParam.nextURL}">
  <c:set var="nextURL" value="{WParam.nextURL}" />
</c:if>
```

9. Press Ctrl+S followed by Ctrl+W to save and close the file.

After you make these modifications, follow steps 2 through 5 in “Test the new pages” on page 286 to test the new pages. Try the following scenarios:














- ▶ Check out as a guest shopper, for example no login at any point
- ▶ Check out as a registered shopper, for example log in before checkout
- ▶ Check out as guest shopper, but log in on the sign in or check out page

Remove the option to pay in store from the payment page

Currently, the option to pay in the store is always included in the payment method page. Because this does not make sense for shoppers that choose to have the products shipped to them, we need to filter out this payment option from the payment method page unless the shopper has selected the pick up in store delivery option.

Attention: In this simplified example, we remove only the option to pay in store from the payment method page. For a product-ready solution, you need to implement additional validation logic in a customized version of the `com.ibm.commerce.order.commands.ValidatePaymentMethodCmd` task command. You need to implement this validation logic to prevent a malicious user from manipulating the request parameters.

To customize the payment method page:

1. Open the development environment by selecting **Start**  **Programs**  **IBM**  **WebSphere**  **WebSphere Commerce**  **WebSphere Commerce Developer Enterprise**.
2. Switch to the Java EE perspective if it is not already active:
 - a. Select **Window**  **Open Perspective**  **Other**.
 - b. The Open Perspective window opens. Select **Java EE**, and click **OK**.
3. In the Enterprise Explorer, expand and double-click **Stores**  **WebContent**  **Madisons**  **mobile**  **ShoppingArea**  **CheckoutSection**  **OrderPaymentDetails.jsp**.
4. The file opens in the JSP source code editor. Locate the code shown in Example 5-47 (reformatted here for readability).

Example 5-47 Payment method selection HTML from OrderPaymentDetails.jsp

```
<!-- Enable Pay in store payment type -->
<c:forEach
    items="${paymentPolicyListDataBean.paymentPolicyInfoUsableWithoutTA}"
    var="paymentPolicyInfo" varStatus="status">
    <c:if test="${ !empty paymentPolicyInfo.attrPageName }" >
        <c:if test="${paymentPolicyInfo.attrPageName == 'StandardPayLater'}">
            <div class="radio_container">
                <input type="radio" checked
                    id="${paymentPolicyInfo.policyName}"
                    name="payMethodId_radio"
                    value="${paymentPolicyInfo.policyName}" />
                <label for="${paymentPolicyInfo.policyName}">
                    <fmt:message key="PAY_IN_STORE" bundle="${storeText}" />
                </label>
            </div>
        </c:if>
    </c:if>
</c:forEach>
```

We need to filter out this code if the shopper has selected the shop online option during checkout. We use the fact that the pay in store payment option is not considered a valid method by the IBM WebSphere Commerce server in that instance. Thus, we enumerate the available payment methods and only emit the above code if pay in store is among the methods.

5. To implement this feature, add the code from Example 5-48 just *before* the first line of code shown in Example 5-47.

Example 5-48 Start of the block to check for the pay in store payment method

```
<%--
    only display payment type selection if one of the
    valid methods include "PayInStore"
--%>
<c:set var="shipMode"
    value="${order.orderItem[0].orderItemShippingInfo.shippingMode}" />
<c:set var="shipModeCode"
    value="${shipMode.shippingModeIdentifier.externalIdentifier.shipModeCode}" />
<c:if test="${shipModeCode == 'PickupInStore'}">
```

6. Add the code from Example 5-49 just *after* the last line of code shown in Example 5-47.

Example 5-49 End of the block to check for the pay in store payment method

```
</c:if>
```

The code in Example 5-49 removes the radio button for the pay in store method if the order is being shipped by showing only the radio button if the current shipping mode corresponds to the standard `PickupInStore` mode.

Because we simply remove one radio button in this example, the layout is slightly odd because the form will have a single radio button that the user has to select in order to pay by credit card. A more complete example removes the radio buttons altogether, but that method requires more substantial changes to the page. For this reason, we chose this simpler approach.














7. Press Ctrl+S followed by Ctrl+W to save and close the `OrderPaymentDetails.jsp` file.

After you make these modifications, follow steps 2 through 12 in “Test the new pages” on page 286 to test the new page, by skip step 6, because the logic for redirecting to the shipping address selection page correctly was added in the previous section. Try the testing with both the shop online and pick up in store scenarios to observe the differences on the payment details page.

Modify the order summary page to show shipping address

The current order summary page, as well as the order confirmation page and e-mail, all assume that the order does not have a shipping address, but rather a store location. Incidentally, the shipping address *is* shown, but it is presented as a store location, as shown in Figure 5-38 on page 290. We use the fact that an order that a shopper has asked to be picked up in a store is associated with a special shipping mode with the `PickupInStore` code.

To customize the order summary page:

1. Open the development environment by selecting **Start**  **Programs**  **IBM**  **WebSphere**  **WebSphere Commerce**  **WebSphere Commerce Developer Enterprise**.
2. Switch to the Java EE perspective if it is not already active:
 - a. Select **Window**  **Open Perspective**  **Other**.
 - b. The Open Perspective window opens. Select **Java EE**, and click **OK**.
3. In the Enterprise Explorer, expand and double-click **Stores**  **WebContent**  **Madisons**  **mobile**  **ShoppingArea**  **OrderSection**  **OrderSummaryDisplay.jsp**.
4. The file opens in the JSP source code editor. Locate the code shown in Example 5-50 (reformatted here for readability).

Example 5-50 Store location display code for OrderSummaryDisplay.jsp

```
<div id="store_location">
  <h3><fmt:message key="MO_STORE_LOCATION" bundle="{storeText}"/></h3>
  <p><fmt:message key="MO_STORE_PICK_UP_MSG" bundle="{storeText}"/>.</p>
  <ul>
    <c:set var="contact"
      value="{order.orderItem[0].orderItemShippingInfo.shippingAddress}" />
    <c:if test="{!empty
      contact.contactInfoIdentifier.externalIdentifier.contactInfoNickName}">
      <li>
        <p><c:out
value="{contact.contactInfoIdentifier.externalIdentifier.contactInfoNickName}"/></p>
        </li>
      </c:if>
      <li>
        <!-- Display shipping address of the order --%>
        <%@ include file="../../Snippets/ReusableObjects/AddressDisplay.jspf"%>
        </li>
      <wcf:url var="StoreURL" value="mSelectedStoreListView">
        <wcf:param name="langId" value="{langId}" />
        <wcf:param name="storeId" value="{WCPParam.storeId}" />
        <wcf:param name="orderId" value="{WCPParam.orderId}" />
        <wcf:param name="fromPage" value="ShoppingCart" />
        <wcf:param name="catalogId" value="{WCPParam.catalogId}" />
      </wcf:url>
      <li>
        <span class="bullet">&#187; </span>
        <a href="{fn:escapeXml(StoreURL)}"
          title="<fmt:message key="MO_STORE_CHANGE_TITLE"
```

```

        bundle="${storeText}"/>"><fmt:message key="MO_STORE_CHANGE"
        bundle="${storeText}"/>
    </a>
</li>
</ul>
</div>

```

5. Insert the code in Example 5-51 just *before* the code block shown in Example 5-50.

Example 5-51 JSP code to check the shipping mode code

```

<c:set var="shipMode"
    value="${order.orderItem[0].orderItemShippingInfo.shippingMode}" />
<c:set var="shipModeCode"
    value="${shipMode.shippingModeIdentifier.externalIdentifier.shipModeCode}" />
<c:choose>
    <c:when test="${shipModeCode == 'PickupInStore'}">

```

6. Insert the code in Example 5-52 just *after* the code block shown in Example 5-50 on page 300.
7. Press Ctrl+S followed by Ctrl+W to save and close the file.

Example 5-52 Remaining code to show the shipping address and method for OrderSummaryDisplay.jsp

```

</c:when>
<c:otherwise>
    <div id="shipping_info">
        <h3><fmt:message key="MO_SHIPPING_ADDRESS" bundle="${storeText}"/></h3>
        <ul>
            <c:set var="contact"
                value="${order.orderItem[0].orderItemShippingInfo.shippingAddress}" />
            <c:set var="contactId"
                value="${contact.contactInfoIdentifier.externalIdentifier}" />
            <c:if test="${!empty contactId.contactInfoNickName}">
                <li>
                    <p><c:out value="${contactId.contactInfoNickName}"/></p>
                </li>
            </c:if>
            <li>
                <!-- Display shipping address of the order --%>
                <%@ include file="../../../Snippets/ReusableObjects/AddressDisplay.jspf"%>
            </li>
            <wcf:url var="ShippingAddressURL" value="mOrderShippingAddressSelection">
                <wcf:param name="langId" value="${langId}" />
                <wcf:param name="storeId" value="${WParam.storeId}" />
                <wcf:param name="catalogId" value="${WParam.catalogId}" />
            </wcf:url>

```

```

<li>
  <span class="bullet">&#187; </span>
  <a href="{fn:escapeXml(ShippingAddressURL)}"
    title="<fmt:message key="MO_EDIT_SHIPPING_ADDR_TITLE"
    bundle="{storeText}"/>"><fmt:message key="MO_EDIT"
    bundle="{storeText}"/>
  </a>
</li>
</ul>

<h3><fmt:message key="SHIPPING_METHOD_SELECTION_TITLE"
  bundle="{storeText}"/></h3>
<ul>
  <wcf:url var="ShippingMethodURL" value="mOrderShippingMethodSelection">
    <wcf:param name="langId" value="{langId}" />
    <wcf:param name="storeId" value="{WCPParam.storeId}" />
    <wcf:param name="catalogId" value="{WCPParam.catalogId}" />
  </wcf:url>
  <li>
    <p><c:out value="{shipModeCode}"/></p>
  </li>
  <li>
    <span class="bullet">&#187; </span>
    <a href="{fn:escapeXml(ShippingMethodURL)}"
      title="<fmt:message key="SHIPPING_METHOD_SELECT"
      bundle="{storeText}"/>"><fmt:message key="MO_EDIT"
      bundle="{storeText}"/></a>
  </li>
</ul>
</div>
</c:otherwise>
</c:choose>

```

Note the similarities between parts of the code in Example 5-52 and the original store location display block from Example 5-51 on page 301. The reason is that the store location is set up as the shipping information when the shopper selects pick up in store. We need to change the header and link information below the address. The latter part is the code to show the selected shipping method and providing a link to modify it.

You can now test the order summary page. Note that because we introduced two new localized texts, these text show as errors on the page as shown in Figure 5-40. We describe how to add these texts in “Define localized strings for customized pages” on page 312.

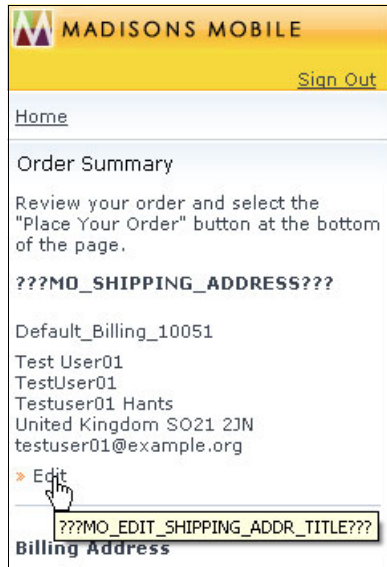















Figure 5-40 Order summary page with shipping address and missing texts

Modify order confirmation page to show shipping address

The change to the order confirmation page is very similar to the change you just did for the order summary page, except for the simplification that there is no link for changing the address. To customize the order confirmation page to show the shipping address header instead of the store location on the order confirmation page:

1. Open the development environment by selecting **Start**  **Programs**  **IBM**  **WebSphere**  **WebSphere Commerce**  **WebSphere Commerce Developer Enterprise**.
2. Switch to the Java EE perspective if it is not already active:
 - a. Select **Window**  **Open Perspective**  **Other**.
 - b. The Open Perspective window opens. Select **Java EE**, and click **OK**.
3. In the Enterprise Explorer, expand and double-click **Stores**  **WebContent**  **Madisons**  **mobile**  **ShoppingArea**  **OrderSection**  **OrderConfirmationDisplay.jsp**.

4. The file opens in the JSP source code editor. Locate the code shown in Example 5-53.

Example 5-53 Store location header

```
<div id="store_location">
  <h3><fmt:message key="MO_STORE_LOCATION" bundle="${storeText}"/></h3>
  <p><fmt:message key="MO_STORE_PICK_UP_MSG" bundle="${storeText}"/>.</p>
```

5. *Replace* the code from Example 5-53 with the code in Example 5-54.

Example 5-54 New code to display either the store location or the shipping address header

```
<c:set var="shipMode"
  value="${order.orderItem[0].orderItemShippingInfo.shippingMode}" />
<c:set var="shipModeCode"
  value="${shipMode.shippingModeIdentifier.externalIdentifier.shipModeCode}" />
<c:choose>
  <c:when test="${shipModeCode == 'PickupInStore'}">
    <div id="shipping_address">
      <h3><fmt:message key="MO_STORE_LOCATION" bundle="${storeText}"/></h3>
      <p><fmt:message key="MO_STORE_PICK_UP_MSG" bundle="${storeText}"/>.</p>
    </c:when>
    <c:otherwise>
      <div id="shipping_address">
        <h3><fmt:message key="MO_SHIPPING_ADDRESS" bundle="${storeText}"/></h3>
      </c:otherwise>
    </c:choose>
```

6. Locate the code shown in Example 5-55.

Example 5-55 End of the shipping <div> and beginning of the billing <div>

```
</div>

<div id="billing_info">
```

7. Insert the code from Example 5-56 *before* the code in Example 5-55, for example, just before the </div> tag.

Example 5-56 New code to display either the store location or the shipping address header

```
<c:if test="${shipModeCode != 'PickupInStore'}">
  <h3><fmt:message key="SHIPPING_METHOD_SELECTION_TITLE"
    bundle="${storeText}"/></h3>
  <ul>
    <li>
      <p><c:out value="${shipModeCode}"/></p>
```

```
</li>
</ul>
</c:if>
```

8. Press Ctrl+S followed by Ctrl+W to save and close the file.

Again, after you make these modifications, follow the directions in “Test the new pages” on page 286 to test the customized page. You need to add the newly introduced localized heading for the shipping address display to the store text resource bundle, as shown in Figure 5-41. We describe how to add this text in “Define localized strings for customized pages” on page 312.

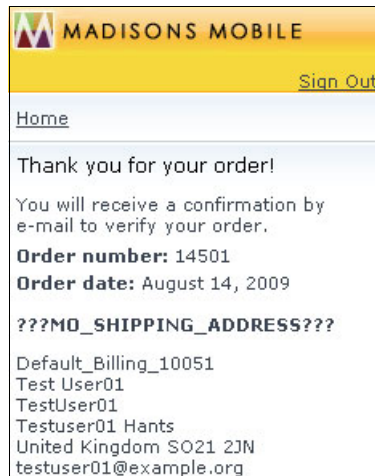


Figure 5-41 Customized order confirmation page with missing localized text













Add breadcrumb support for new pages and flow

The breadcrumb paths in the Madisons mobile starter store are generated by a central JSP fragment, `BreadCrumbTrailDisplay.jspf`, that is included from all pages that need to display a breadcrumb. If you tested the pages that you added and modified so far, you might have noticed that the breadcrumb trail is either missing or incorrect for the shop online scenario. This section explains how to add breadcrumb support for new pages and flow.

Add support for shop online to `BreadCrumbtrailDisplay.jspf`

In this section, we explain how to modify the breadcrumb JSP fragment to support the new pages that were added in 5.4.5, “Create new pages” on page 241, as well as the overall changes to the page flow.

To modify the breadcrumb JSP fragment:

1. Open the development environment by selecting **Start**  **Programs**  **IBM**  **WebSphere**  **WebSphere Commerce**  **WebSphere Commerce Developer Enterprise**.
2. Switch to the Java EE perspective if it is not already active:
 - a. Select **Window**  **Open Perspective**  **Other**.
 - b. The Open Perspective window opens. Select **Java EE**, and click **OK**.
3. In the Enterprise Explorer, expand and double-click **Stores**  **WebContent**  **Madisons**  **mobile**  **include**  **BreadCrumbTrailDisplay.jspf**.
4. The file opens in the JSP source code editor. Locate the code shown in Example 5-57.

Example 5-57 Billing address selection URL variable from BreadCrumbTrailDisplay.jspf

```
<wcf:url var="BillingSelectionURL" value="mOrderBillingAddressSelection">
  <wcf:param name="langId" value="{langId}" />
  <wcf:param name="storeId" value="{WParam.storeId}" />
  <wcf:param name="catalogId" value="{WParam.catalogId}" />
</wcf:url>
```

5. Insert the code from Example 5-58 *before* the code shown in Example 5-57.

The code in Example 5-58 defines the targets for the shipping address selection and shipping address editing pages and is used in further code fragments to generate links back to these pages from further along the checkout flow.

Example 5-58 Shipping address selection and editing URL variables for BreadCrumbTrailDisplay.jspf

```
<wcf:url var="ShippingAddressSelectionURL" value="mOrderShippingAddressSelection">
  <wcf:param name="langId" value="{langId}" />
  <wcf:param name="storeId" value="{WParam.storeId}" />
  <wcf:param name="catalogId" value="{WParam.catalogId}" />
</wcf:url>
<wcf:url var="ShippingMethodSelectionURL" value="mOrderShippingMethodSelection">
  <wcf:param name="langId" value="{langId}" />
  <wcf:param name="storeId" value="{WParam.storeId}" />
  <wcf:param name="catalogId" value="{WParam.catalogId}" />
</wcf:url>
```

6. Locate the code shown in Example 5-59 in the `BreadCrumbTrailDisplay.jspf` file.

Example 5-59 Code to generate breadcrumb for the sign in or check out page

```
<c:when test="${checkoutLogonPage}"> <!-- Checkout logon page --%>
  <fmt:message key="DIVIDING_BAR" bundle="${storeText}" />
  <a href="${fn:escapeXml(ShoppingCartURL)}"><fmt:message key="BCT_SHOPPINGCART"
bundle="${storeText}" /></a>
  <fmt:message key="DIVIDING_BAR" bundle="${storeText}" />
  <span class="current_page"><fmt:message key="BCT_CHECKOUTLOGON"
bundle="${storeText}" /></span>
</c:when>
```

7. Insert the code from Example 5-60 *after* the code shown in Example 5-59.

The code in Example 5-60 generates the breadcrumb paths for the new pages in the checkout flow:

- Shipping address selection
- Shipping address create/edit
- Shipping method selection
- Shipping instructions

Example 5-60 Code to generate bread crumbs for the new checkout pages

```
<c:when test="${shippingSelectionPage}">
  <fmt:message key="DIVIDING_BAR" bundle="${storeText}" />
  <a href="${fn:escapeXml(ShoppingCartURL)}"><fmt:message key="BCT_SHOPPINGCART"
bundle="${storeText}" /></a>
  <fmt:message key="DIVIDING_BAR" bundle="${storeText}" />
  <span class="current_page"><fmt:message key="SHIPPING_ADDRESS_SELECT_TITLE"
bundle="${storeText}" /></span>
</c:when>
```

```
<c:when test="${shippingDetailsPage}">
  <fmt:message key="DIVIDING_BAR" bundle="${storeText}" />
  <a href="${fn:escapeXml(ShoppingCartURL)}"><fmt:message key="BCT_SHOPPINGCART"
bundle="${storeText}" /></a>
  <fmt:message key="DIVIDING_BAR" bundle="${storeText}" />
  <a href="${fn:escapeXml(ShippingAddressSelectionURL)}"><fmt:message
key="SHIPPING_ADDRESS_SELECT_TITLE" bundle="${storeText}" /></a>
  <fmt:message key="DIVIDING_BAR" bundle="${storeText}" />
  <span class="current_page"><fmt:message key="SHIPPING_ADDRESS_DETAILS_TITLE"
bundle="${storeText}" /></span>
</c:when>
```

```

<c:when test="${shippingMethodSelectionPage}">
    <fmt:message key="DIVIDING_BAR" bundle="${storeText}" />
    <a href="${fn:escapeXml(ShoppingCartURL)}"><fmt:message key="BCT_SHOPPINGCART"
bundle="${storeText}" /></a>
    <fmt:message key="DIVIDING_BAR" bundle="${storeText}" />
    <a href="${fn:escapeXml(ShippingAddressSelectionURL)}"><fmt:message
key="SHIPPING_ADDRESS_SELECT_TITLE" bundle="${storeText}" /></a>
    <fmt:message key="DIVIDING_BAR" bundle="${storeText}" />
    <span class="current_page"><fmt:message key="SHIPPING_METHOD_SELECTION_TITLE"
bundle="${storeText}" /></span>
</c:when>

<c:when test="${shippingInstructionsPage}">
    <fmt:message key="DIVIDING_BAR" bundle="${storeText}" />
    <a href="${fn:escapeXml(ShoppingCartURL)}"><fmt:message key="BCT_SHOPPINGCART"
bundle="${storeText}" /></a>
    <fmt:message key="DIVIDING_BAR" bundle="${storeText}" />
    <a href="${fn:escapeXml(ShippingAddressSelectionURL)}"><fmt:message
key="SHIPPING_ADDRESS_SELECT_TITLE" bundle="${storeText}" /></a>
    <fmt:message key="DIVIDING_BAR" bundle="${storeText}" />
    <a href="${fn:escapeXml(ShippingMethodSelectionURL)}"><fmt:message
key="SHIPPING_METHOD_SELECTION_TITLE" bundle="${storeText}" /></a>
    <fmt:message key="DIVIDING_BAR" bundle="${storeText}" />
    <span class="current_page"><fmt:message key="SHIPPING_INSTRUCTIONS_TITLE"
bundle="${storeText}" /></span>
</c:when>

```

8. Locate the code shown in Example 5-61 in the `BreadCrumbTrailDisplay.jspf` file.

Example 5-61 Code to generate breadcrumb for billing address, payment, and order summary pages

```

<c:when test="${billingSelectionPage}">
    <fmt:message key="DIVIDING_BAR" bundle="${storeText}" />
    <a href="${fn:escapeXml(ShoppingCartURL)}"><fmt:message key="BCT_SHOPPINGCART"
bundle="${storeText}" /></a>
    <fmt:message key="DIVIDING_BAR" bundle="${storeText}" />
    <a href="${fn:escapeXml(StoreSelectionURL)}"><fmt:message key="BCT_STORESELECTION"
bundle="${storeText}" /></a>
    <fmt:message key="DIVIDING_BAR" bundle="${storeText}" />
    <span class="current_page"><fmt:message key="BCT_BILLINGSELECTION"
bundle="${storeText}" /></span>
</c:when>

```

```

<c:when test="${billingDetailsPage}">
  <fmt:message key="DIVIDING_BAR" bundle="${storeText}" />
  <a href="${fn:escapeXml(ShoppingCartURL)}"><fmt:message key="BCT_SHOPPINGCART"
bundle="${storeText}" /></a>
  <fmt:message key="DIVIDING_BAR" bundle="${storeText}" />
  <a href="${fn:escapeXml(StoreSelectionURL)}"><fmt:message key="BCT_STORESELECTION"
bundle="${storeText}" /></a>
  <fmt:message key="DIVIDING_BAR" bundle="${storeText}" />
  <a href="${fn:escapeXml(BillingSelectionURL)}"><fmt:message
key="BCT BILLINGSELECTION" bundle="${storeText}" /></a>
  <fmt:message key="DIVIDING_BAR" bundle="${storeText}" />
  <span class="current_page"><fmt:message key="BCT BILLINGDETAILS"
bundle="${storeText}" /></span>
</c:when>

<c:when test="${paymentSelectionPage}">
  <fmt:message key="DIVIDING_BAR" bundle="${storeText}" />
  <a href="${fn:escapeXml(ShoppingCartURL)}"><fmt:message key="BCT_SHOPPINGCART"
bundle="${storeText}" /></a>
  <fmt:message key="DIVIDING_BAR" bundle="${storeText}" />
  <a href="${fn:escapeXml(StoreSelectionURL)}"><fmt:message key="BCT_STORESELECTION"
bundle="${storeText}" /></a>
  <fmt:message key="DIVIDING_BAR" bundle="${storeText}" />
  <a href="${fn:escapeXml(BillingSelectionURL)}"><fmt:message
key="BCT BILLINGSELECTION" bundle="${storeText}" /></a>
  <fmt:message key="DIVIDING_BAR" bundle="${storeText}" />
  <span class="current_page"><fmt:message key="BCT_PAYMENTSELECTION"
bundle="${storeText}" /></span>
</c:when>

<c:when test="${orderSummaryPage}">
  <fmt:message key="DIVIDING_BAR" bundle="${storeText}" />
  <a href="${fn:escapeXml(ShoppingCartURL)}"><fmt:message key="BCT_SHOPPINGCART"
bundle="${storeText}" /></a>
  <fmt:message key="DIVIDING_BAR" bundle="${storeText}" />
  <a href="${fn:escapeXml(StoreSelectionURL)}"><fmt:message key="BCT_STORESELECTION"
bundle="${storeText}" /></a>
  <fmt:message key="DIVIDING_BAR" bundle="${storeText}" />
  <a href="${fn:escapeXml(BillingSelectionURL)}"><fmt:message
key="BCT BILLINGSELECTION" bundle="${storeText}" /></a>
  <fmt:message key="DIVIDING_BAR" bundle="${storeText}" />
  <a href="${fn:escapeXml(PaymentSelectionURL)}"><fmt:message
key="BCT_PAYMENTSELECTION" bundle="${storeText}" /></a>
  <fmt:message key="DIVIDING_BAR" bundle="${storeText}" />

```

```
<span class="current_page"><fmt:message key="BCT_ORDERSUMMARY"
bundle="\${storeText}" /></span>
</c:when>
```

9. Replace *each* of the *four* lines that are highlighted in bold font in Example 5-61 with the code shown in Example 5-62.

The highlighted lines in Example 5-61 generate the link to the store selection page, and the code in Example 5-62 generates either the store selection page link or links to the shipping address and shipping method pages, depending on the shipping mode configured for the order.

Example 5-62 Code to generate breadcrumb links for the shipping pages

```
<c:choose>
  <c:when test="\${WCPParam.fromPage == 'ShoppingCart' ||
order.orderItem[0].orderItemShippingInfo.shippingMode.shippingModeIdentifier.external
Identifier.shipModeCode == 'PickupInStore'}">
    <a href="\${fn:escapeXml(StoreSelectionURL)}"><fmt:message
key="BCT_STORESELECTION" bundle="\${storeText}" /></a>
  </c:when>
  <c:otherwise>
    <a href="\${fn:escapeXml(ShippingAddressSelectionURL)}"><fmt:message
key="SHIPPING_ADDRESS_SELECT_TITLE" bundle="\${storeText}" /></a>
    <fmt:message key="DIVIDING_BAR" bundle="\${storeText}" />
    <a href="\${fn:escapeXml(ShippingMethodSelectionURL)}"><fmt:message
key="SHIPPING_METHOD_SELECTION_TITLE" bundle="\${storeText}" /></a>
  </c:otherwise>
</c:choose>
```










10. Press Ctrl+S followed by Ctrl+W to save and close the file.

The conditional code in Example 5-62 shows the pickup in store breadcrumb if the selected shipping mode is the special PickupInStore or if the request parameter fromPage has the value ShoppingCart. You need to ensure that this request parameter is set for the pickup in store scenario in those cases where the shipping mode is not yet set.

Add breadcrumb support in SaveStoreSelection.jsp

The SaveStoreSelection.jsp file is the page that presents the preferred store list and allows the shopper to select the store from which to pick up. We add the fromPage request parameter to the form that redirects to the billing address selection page to ensure that the breadcrumb shows up correctly on the billing address selection page.

To add the `fromPage` parameter to the `SaveStoreSelection.jsp` file:

1. Open the development environment by selecting **Start**  **Programs**  **WebSphere**  **WebSphere Commerce Developer Enterprise**.
2. Switch to the Java EE perspective if it is not already active:
 - a. Select **Window**  **Open Perspective** .
 - b. The Open Perspective window opens. Select **Java EE**, and click **OK**.
3. In the Enterprise Explorer, expand and double-click **Stores**  **WebContent**  **Madisons**  **StoreLocatorArea** .
4. The file opens in the JSP source code editor. Locate the code shown in Example 5-63.

Example 5-63 Code to generate billing address select URL in `SaveStoreSelection.jsp`

```
<wcf:url var="updateItemPhysicalStore" value="OrderChangeServiceItemUpdate">
  <wcf:param name="langId" value="{langId}" />
  <wcf:param name="storeId" value="{WCPParam.storeId}" />
  <wcf:param name="catalogId" value="{WCPParam.catalogId}" />
  <wcf:param name="orderId" value="." />
  <wcf:param name="URL" value="{WCPParam.refUrl}" />
```

5. Insert the code from Example 5-64 *after* the code shown in Example 5-63.

Example 5-64 Code to generate the `fromPage` parameter

```
<wcf:param name="fromPage" value="{WCPParam.fromPage}" />
```

The code fragment now looks similar to that shown in Example 5-65 with the inserted line highlighted.

Example 5-65 Code to generate billing address select URL with `fromPage` for `SaveStoreSelection.jsp`

```
<wcf:url var="updateItemPhysicalStore" value="OrderChangeServiceItemUpdate">
  <wcf:param name="langId" value="{langId}" />
  <wcf:param name="storeId" value="{WCPParam.storeId}" />
  <wcf:param name="catalogId" value="{WCPParam.catalogId}" />
  <wcf:param name="orderId" value="." />
  <wcf:param name="URL" value="{WCPParam.refUrl}" />
  <wcf:param name="fromPage" value="{WCPParam.fromPage}" />
```

6. Save and close the file by pressing **Ctrl+S** followed by **Ctrl+W**.

Define localized strings for customized pages

As you saw when you tested the customized order summary and confirmation pages, you need to define a few more localized texts to the store text resource bundle. To add these texts:

1. Open the development environment by selecting **Start** ⌵ **Programs** ⌵ **IBM** ⌵ **WebSphere** ⌵ **WebSphere Commerce** ⌵ **WebSphere Commerce Developer Enterprise**.
2. Switch to the Java EE perspective if it is not already active:
 - a. Select **Window** ⌵ **Open Perspective** ⌵ **Other**.
 - b. The Open Perspective window opens. Select **Java EE**, and click **OK**.
3. In the Enterprise Explorer, expand and double-click **Stores** ⌵ **Java Resources: JavaSource** ⌵ **Madisons.mobile** ⌵ **storetext.properties**.
4. The file opens. Insert the properties from Example 5-66 at the *end* of the file, and press Ctrl+S and then Ctrl+W to save and close the file.

Example 5-66 Localized strings for the customized order summary and confirmation pages

```
MO_SHIPPING_ADDRESS = Shipping Address
MO_EDIT_SHIPPING_ADDR_TITLE = Edit shipping address
```

Now that the final texts are added, you can test the entire flow.

Note: You need to restart the server to pick up the changed resource bundle.

5.5 Product ratings and review

As mentioned in “Functions not included in the Madisons Mobile Starter Store” on page 51, the standard mobile pages that are delivered with the Madisons Mobile Starter Store do not contain hooks for the integration with third-party social commerce service providers as the standard Madisons Starter Store does. In this section, we explore how you can extend one of these features, the product ratings and reviews integration component that was introduced in 2.2, “Social Commerce” on page 52, to be used on a mobile device.

We do not describe the steps to install and configure the Social Commerce feature of IBM WebSphere Commerce V7. Refer to the WebSphere Commerce Version 7 Information Center for detailed steps about installation and configuration:

<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.install.doc/tasks/tigsocdev.htm>

Furthermore, to use the code in this section, you need access to a Ratings and Reviews service provider, such as BazaarVoice.

5.5.1 Ratings and reviews on the mobile device

In this section, we discuss the social commerce features that you can add to the Madisons Mobile Starter Store pages. As mentioned previously, we limit the scope to ratings and reviews, which we implement on the following pages:

- ▶ Product display page

On this page, we add the overall average rating, illustrated with a number of stars, as well as a link to the actual reviews, should the shopper want to read the individual reviews. Figure 5-42 shows this page.

Note: The product display page actually consists of four different pages:

- ▶ Product display
- ▶ Item display
- ▶ Bundle display
- ▶ Package display

Thus, you need to update all four of these pages to add the overall ratings to the product page.


MADISON'S MOBILE
[Sign In](#)

[Home](#) | [Furniture](#) | Mahogany Desk Chair

Mahogany Desk Chair

SKU: FUOF-01



Price: **\$249.99**

Qty:

[Add to Cart](#)

Overall Rating ★★☆☆☆

» [Read Reviews \(3\)](#)

» [Add to Wish List](#)

» [Select to Compare](#)

Check Availability:

Online:

✓ In Stock

In-store:

» [Select Store\(s\)](#)

Figure 5-42 The mobile product page with overall product rating added

► Product compare page

Similar to the product display page, you can display the overall rating with a link to the individual reviews for each product shown on the product compare page. Figure 5-43 shows this page.

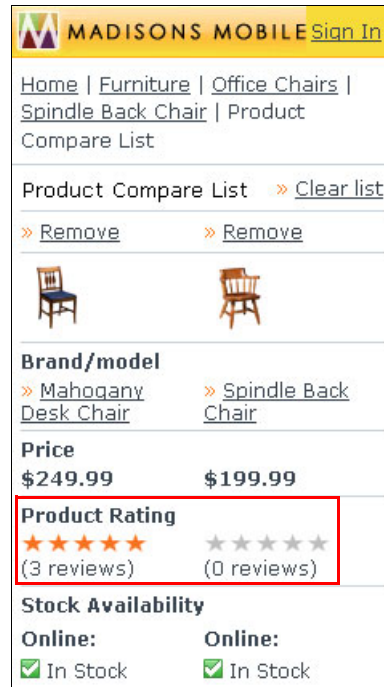


Figure 5-43 Mobile product compare with ratings

► Review list

The review list page is a new page that shows the summary information for all of the ratings for a given product, along with a link to the full review:

- Rating
- Title
- User name
- Review date

Figure 5-44 shows this page.

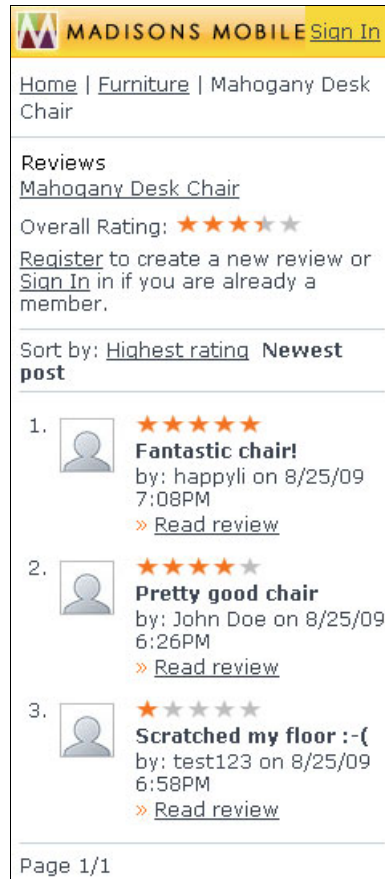


Figure 5-44 Mobile overview of reviews for a given product

► Review details

The review details page is a new page that presents the full details for a given review. Figure 5-45 shows this page.



Figure 5-45 Mobile review details page

► Write a review

The Write a Review page is a new page that allows the user to write a new review. Figure 5-46 shows this page.

Figure 5-46 Mobile write review page

This overview shows that most of the ratings and review pages include the following common elements:

- The rating, summarized as a list of fully or partially filled out stars.
- A button to write a new review, replaced with links to log in or register if the current user is not registered.
- A header with the rating, review title, reviewer and the date and time of the review (common for the review list and review detail pages).

We consolidate the code for these fragments in JSP snippets that are included statically and dynamically to minimize the amount of duplication.

In the following sections, we describe the overall technical design for the new pages and the detailed steps for implementing these modifications.

5.5.2 Integration approach

The standard Social Commerce implementation, as described in 2.2.6, “Social Commerce integration” on page 58, makes use of Ajax to retrieve ratings and reviews from the Social Commerce (Soccom) application within the shopper’s browser, as illustrated in Figure 2-17 on page 59. Due to the limited nature of mobile devices, we want to avoid heavy use of JavaScript for the purpose of retrieving ratings and reviews. For this reason, we implement the ratings and reviews on the mobile device by retrieving the data from the IBM WebSphere Commerce server before the page is served to the shopper.

Note: We are *not* recommending that you avoid JavaScript for mobile devices. The standard pages for the Madisons mobile starter store uses JavaScript to control much of the functionality of the checkout process. However, we have chosen a JavaScript-light approach in this section to highlight how you can develop mobile pages that implement Social Commerce without relying on JavaScript libraries.

Figure 5-47 and Figure 5-48 illustrate the different approaches.

Figure 5-47 illustrates the sequence of invocations for the Ajax approach. The page is returned to the browser, and then from the original request the browser retrieves the review data and renders the results asynchronously.

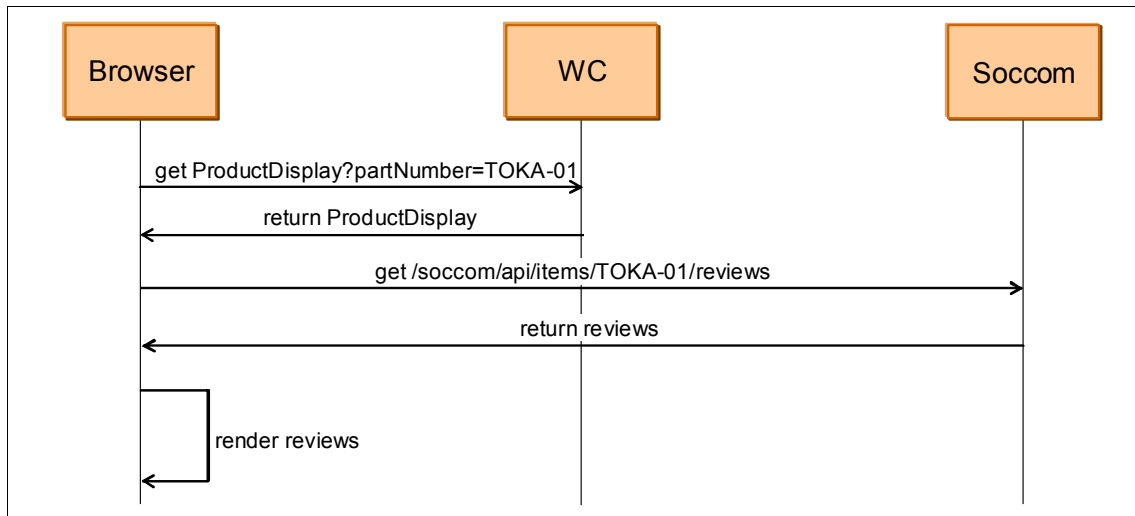


Figure 5-47 In-browser AJAX integration to the Social Commerce (Soccom) application

In contrast, Figure 5-48 illustrates the sequence of invocations for the in-JSP integration approach, where the page is not returned to the browser until the data is retrieved from the Social Commerce server and rendered in the JSP itself.

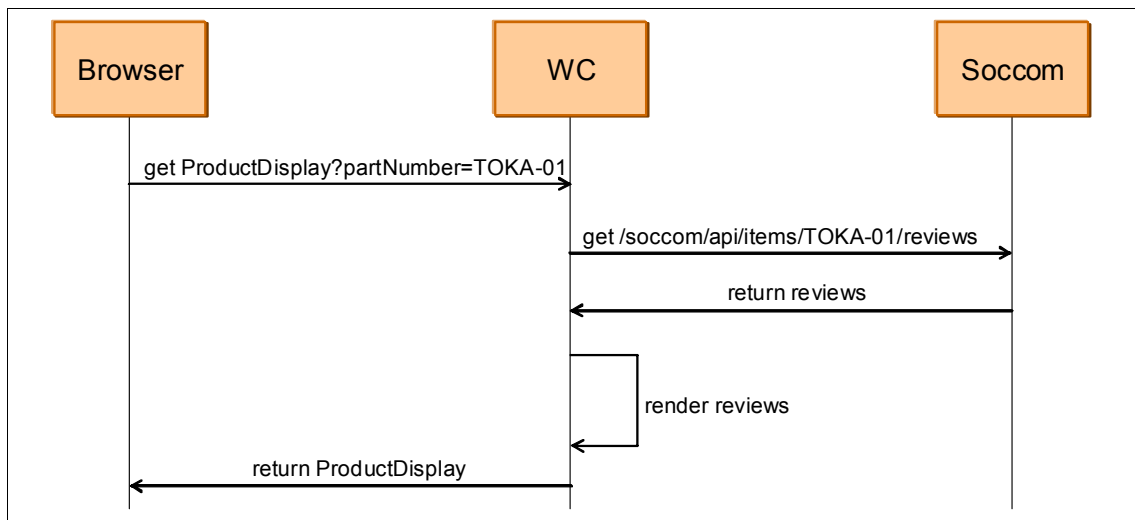


Figure 5-48 In-JSP integration to social commerce (Soccom) application

To implement this scenario, you need to complete these steps:

1. Prepare the workspace.

Before you can use the code for the social commerce widgets, you need to prepare your IBM WebSphere Commerce V7 Developer Edition workspace.

2. Create the ratings and reviews data beans.

TO encapsulate the integration code that is necessary to retrieve product reviews from the Soccom application, you need to develop a data bean that can be used from the JSPs to populate the ratings and review information for a given product.

3. Create the generic star display JSP.

Because all of the pages that you will create or modify need to display the rating using a number of stars, you need to create a JSP that can be included on the relevant pages whenever stars are needed to illustrate a given rating.

4. Create the JSP fragment for the Write Review button.

In recognition that several pages need logic to either display a button for writing a review or display sign in or log on links, you need to isolate this logic in a JSP snippet that is included statically.

5. Create the JSP fragment for the review header.

The review header with the avatar image, review title, reviewer's user ID, and review date and time is used on both the review list and the review details page. Thus, you need to create a JSP snippet, which is included statically, that displays this information.

6. Modify the product display pages.

You need add the overall rating with a link to the rating details to all of the product display pages.

7. Modify the product compare page.

Because a major part of the buying decision for a product comparison is the overall rating, you also need to add the rating to the product compare page.

8. Create the JSP for the review list and overview.

Then, you need to create the page that displays the rating overview.

9. Create a page for review details.

You need to create the review details that show a single full review.

10. Create the JSP for writing the review.

You then create the page for submitting a new review from a mobile device.

11. Create the command to post the review.

You need to create the command for submitting a new review to the social commerce application.

12. Modify infrastructure resources.

Finally, you need to modify dependent resources to support the pages and commands that you created.

The following sections describe these steps in detail.

5.5.3 Prepare the workspace



You use external libraries to deserialize JSON objects and to communicate with the Social Commerce application. As such, you need to prepare the IBM WebSphere Commerce V7 Developer Edition workspace with these libraries.

Add the JSON library

You need a library to deserialize the JSON objects that are returned by the Social Commerce application. We used the reference implementation that is available for download from [json.org](http://www.json.org) at:

<http://www.json.org/java/json.zip>

This code is only source code, so you need to compile the code and repackage it as a JAR file. We did this by importing the source code into a new Java project and exporting the JAR file as follows:

1. Download the [json.org](http://www.json.org) source code, and save the compressed file in a temporary location.
2. Switch to the Java perspective of IBM WebSphere Commerce V7 Developer Edition.
3. Select **File**  **New**  **Java Project**.
4. In the Project name field of the New Java Project window that opens, enter [json.org](http://www.json.org), and click **Finish**.
5. Expand **json.org** and select **src**. Then, select **Import**.
6. In the Import window, expand and select **General**  **Archive File**, and click **Next**.
7. On the second page of the Import wizard, enter the location of the [json.zip](http://www.json.org) file that you downloaded in step 1 in the URL field and click **Finish**.
8. In the Package Explorer, right-click **json.org**, and select **Export**.
9. In the Export window, expand and select **Java**  **JAR file**, and click **Next**.

10. On the second page of the Export wizard, clear the following locations:

- .classpath
- .project

Then, expand **json.org** and select **src**. Clear the **test** option.

11. Ensure that the “Export generated class files and resources” option is select, enter the following file name in the JAR file entry field, and click **Finish**:

WC\lib\json.jar

Add the Apache HttpClient V4.0.1 libraries

You use the Apache HttpClient V4.0 library to communicate with the Social Commerce application. Download the following files from the Apache HttpComponents Web site:

- ▶ httpcomponents-client-4.0.1-bin.zip
- ▶ httpcomponents-core-4.0.1-bin.zip

You can find these files by selecting the Download link from the following Web site:

<http://hc.apache.org/>

After downloading these files, expand the archives to a temporary location and copy the files in the lib directories to *WC_Home\workspace\WC\lib*. Copy the following files:

- ▶ httpclient-4.0.jar
- ▶ httpmime-4.0.jar
- ▶ httpcore-4.0.1.jar
- ▶ httpcore-nio-4.0.1.jar

Add the Social Commerce feature

We assume that you enabled the Social Commerce features in your development environment, following these instructions in the information center:

<https://jtcid.hursley.ibm.com/commerce/topic/com.ibm.commerce.install.doc/tasks/tigsoccom.htm>

These instructions create three new projects in the workspace:

- ▶ SocApp
- ▶ soccom
- ▶ SocCore

5.5.4 Create the ratings and reviews data beans

As mentioned previously, you encapsulate the logic for retrieving the review overview and details from the Social Commerce application within a *data bean*, which enables JSP developers to use JSTL tags to instantiate and populate review details within the JSPs themselves.

In our scenario, we develop the following beans:

- ▶ A smart data bean, `ReviewsDataBean`, which is instantiated on the JSPs that use the `<wcf:useBean>` tag.
- ▶ A value object, `ReviewDetailBean`, which holds details of each review. This bean is returned as a `LinkedHashMap` from `ReviewsDataBean`. The individual values in this value object can be accessed through getter and setter methods.

We provide details about these beans in the following sections.

Create the `ReviewsDataBean` class

This data bean extends `SmartDataBeanImpl` and is used to retrieve the review list and details for an item or product by querying the sMash application. The main responsibilities of the bean are to send the request to the sMash application and then parse the response. The request is submitted as an HTTP request, and the response is a JSON object. If the HTTP request is a POST request, the post data contains a JSON object that details the request.

For more information, refer to the *REST API* topic in the WebSphere Commerce Version 7 Information Center at:

<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/topic/com.ibm.commerce.socom.doc/refs/rmsocrestapi.htm>

For retrieving a list of reviews, you need to send a GET request to the URL:

`/items/itemID/reviews`

Where *itemID* is a unique reference to the SKU for which you want to retrieve the reviews. The response is a JSON object of the structure outlined in Example 5-67.

Example 5-67 Review list response syntax

```
{
  "status": { "type": "integer" },
  "message": { "type": "string" },
  "data": {
    "id": { "type": "string" },
```

```

        "ratingStatistics":{ "type": "ratingStatistics" },
        "paginationDetails":{ "type":"paginationDetails" },
        "reviews":{ "type": "array", "arrayType": "review" }
    }
}

```

Example 5-68 shows a sample response for review list query from the Social Commerce application.

Example 5-68 Review list response

```

{"status":200,"message":"","data":{"ratingStatistics":{"ratingRange":{"upperBound":5,"lowerBound":0},"numOfOverallRatings":1,"avgOverallRating":5.0},"paginationDetails":{"sortDirection":"desc","noOfEntriesPerPage":10,"noOfTotalEntries":1,"pageNum":1,"sortType":"default","noOfTotalPages":1},"reviews":[{"reviewId":1,"rating":5,"itemId":"FUL0-01","ItemDescription":"FUL0-01","reviewText":"This is a test review","title":"test review","isRatingOnly":"false","userId":"wcsadmin","recommend":"true","userid":"wcsadmin","submissionTime":"2009-08-26T18:57:56.031"}],"id":"FUL0-01"}}

```

To implement the ReviewsDataBean:

1. Create a new package, `com.xxx.commerce.socom.beans`, where `xxx` is the name of your company in the WebSphereCommerceServerExtensionsLogic project.

Note: The chosen package name is just an example. The name can be anything per your guidelines. For our development, we used the name `com.ibm.itso.commerce.socom.beans`.

2. Create a new class, `ReviewsDataBean`, under this created package. This data bean extends `com.ibm.commerce.beans.SmartDataBeanImpl` and implements `com.ibm.commerce.beans.SmartDataBean`.
3. Add the constructor and attributes to the class as shown in Example 5-69.

The attributes that you add are the input parameters that are required by the sMash application and the output elements that are returned from sMash as a result of fetching the reviews.

In addition to the attributes and constructor, the code in Example 5-69 also contains utility methods for setting and querying the bean's error state.

Note: Some of the types in Example 5-69 refer to packages that you have not yet imported. As such, you will see some compilation errors in the Problems view. We explain how to add the package import statements and resolve these errors in a later step.

Example 5-69 Attributes and constructor for ReviewsDataBean

```
/** constant for specifying a non-initialized integer */
private static final int INT_NO_SET = -1;
/** sets the number of reviews to be displayed on a page */
private int pageSize = INT_NO_SET;
/** sets the page number */
private int onPage = INT_NO_SET;
/** sets the sort type & sort direction of the results.
 * Its format is sort type|sort direction. Valid values for sort
 * type are "rating" and "submissionTime". Valid values for sort
 * direction are "asc" & "desc". */
private String sortOptions;
/** sets the Part number for which results are needed. */
private String partNumber;
/** total number of ratings for the product. */
private int numOfOverallRatings;
/** this is average overall rating for the product. */
private double avgOverallRating;
/** Lower bound used to calculate the average rating. */
private int lowerBound;
/** Upper bound used to calculate the average rating */
private int upperBound;
/** Sorted map of reviewId and ReviewDetailBean */
private LinkedHashMap<String, ReviewDetailBean> reviewDetailsMap;
/** Sort direction used for sorting */
private String sortDirection;
/** Number of results returned by sMash application, to be
 * displayed on a page. - used in Pagination */
private int noOfEntriesPerPage;
/** Total number of reviews for the product */
private int noOfTotalEntries;
/** Current page number - used in pagination */
private int pageNum;
/** Sort type used for sorting */
private String sortType;
/** Number of total pages - Used in pagination */
private int noOfTotalPages;
/** The error message for display */
```

```

private String errorMessage = null;
/** Any exception data, No error if <code>null</code> */
private Throwable errorException = null;
/** Constant for use in tracing and error handling */
private static final String CLASS_NAME =
    ReviewsDataBean.class.getName();
/** Logging and tracing instance */
private static final Logger LOGGER =
    Logger.getLogger(CLASS_NAME);
/** Constructor. Will only initialize the reviewDetailsMap */
public ReviewsDataBean() {
    reviewDetailsMap =
        new LinkedHashMap<String, ReviewDetailBean>();
}
/** Set the error message and optional exception in one call.
    <code>t</code> may be <code>null</code> */
private void setErrorState(String error, Throwable t) {
    errorMessage = error;
    errorException = t;
}
/** Return whether an error has been reported */
public boolean isError() {
    return errorMessage != null || errorException != null;
}

```

4. Add accessor methods for these variables. Add setter methods for all the inputs parameters and getter methods for all the output elements. Add the accessor methods from Example 5-70. (The code in Example 5-70 is shown in compact form.) Press Ctrl+Shift+F to reformat the source code if you want.

Example 5-70 Accessor methods for ReviewsDataBean

```

public String getErrorMessage() {return errorMessage;}
public Throwable getErrorException() {return errorException;}
public int getNumOfOverallRatings() {return numOfOverallRatings;}
public double getAvgOverallRating() {return avgOverallRating;}
public int getLowerBound() {return lowerBound;}
public int getUpperBound() {return upperBound;}
public LinkedHashMap<String, ReviewDetailBean> getReviewDetailsMap()
{return reviewDetailsMap;}
public String getSortDirection() {return sortDirection;}
public int getNoOfEntriesPerPage() {return noOfEntriesPerPage;}
public int getNoOfTotalEntries() {return noOfTotalEntries;}
public int getPageNum() {return pageNum;}
public String getSortType() {return sortType;}

```

```

public int getNoOfTotalPages() {return noOfTotalPages;}
public void setPageSize(int pageSize) {this.pageSize = pageSize;}
public void setOnPage(int onPage) {this.onPage = onPage;}
public void setSortOptions(String sortOptions) {this.sortOptions =
sortOptions;}
public void setPartNumber(String partNumber) {this.partNumber =
partNumber;}

```

5. Add the populate method shown in Example 5-71 to the ReviewsDataBean class. The populate method mainly performs tracing and error handling, referring the request and response handling to the executeRequest and handleResponse methods, which you will add later.

Example 5-71 Populate method of ReviewsDataBean

```

public void populate() {
    final String METHOD_NAME = "populate";
    LOGGER.entering(CLASS_NAME, METHOD_NAME);
    String response = "";
    try {
        if (LOGGER.isLoggable(Level.FINE)) {
            LOGGER.logp(
                Level.FINE, CLASS_NAME, METHOD_NAME,
                "partNumber: \"{0}\"", pageSize: \"{1}\"", "+
                "onPage: \"{2}\"", sortOptions: \"{3}\"",
                new String[] {
                    partNumber, String.valueOf(pageSize),
                    String.valueOf(onPage), sortOptions });
        }
        if (partNumber != null) {
            response = executeRequest();
        }
        else {
            // signal missing parameter with an empty exception,
            // just to keep it non-null
            String message = "Missing parameter \"partNumber\"";
            LOGGER.logp(Level.SEVERE, CLASS_NAME, METHOD_NAME,
                message);
            setErrorState(message, new Exception(message));
        }
    } catch (Exception e) {
        String message =
            "Exception in trying to retrieve review details";
        LOGGER.logp(Level.SEVERE, CLASS_NAME, METHOD_NAME,
            message, e);
    }
}

```

```

        setErrorState(message, e);
    }
    if (response != null && !response.equals("")) {
        handleResponse(response);
    }
    LOGGER.exiting(CLASS_NAME, METHOD_NAME);
}

```

6. Add the `executeRequest` method as shown in Example 5-72 to the `ReviewsDataBean` class. This method creates the necessary `HttpClient` objects to open and execute an HTTP request against the Social Commerce application, passing in the parameters set by the calling JSP as HTTP GET parameters.

The method then retrieves the response from the Social Commerce application as a `String` object if the HTTP response code is in the 200 to 400 range, for example not a known HTTP error code. Otherwise, an error state variable is set.

Example 5-72 Source code for the `executeRequest` method

```

private String executeRequest()
    throws URISyntaxException, IOException,
           ClientProtocolException {

    final String METHOD_NAME = "executeRequest";
    LOGGER.entering(CLASS_NAME, METHOD_NAME);
    String response = "";
    HttpClient client = new DefaultHttpClient();
    //TODO: make socket timeout configurable
    client.getParams().setParameter("http.socket.timeout", 5000);
    client.getParams().setParameter(
        "http.protocol.content-charset", "utf-8");

    List<NameValuePair> params = new LinkedList<NameValuePair>();
    if (pageSize != INT_NO_SET) {
        params.add(new BasicNameValuePair("pageSize",
            String.valueOf(pageSize)));
    }
    if (onPage != INT_NO_SET) {
        params.add(new BasicNameValuePair("onPage",
            String.valueOf(onPage)));
    }
    params.add(new BasicNameValuePair("sortOptions", sortOptions));

    //TODO: make request URI values configurable
}

```

```

String urlScheme = "http";
String urlHost = "localhost";
int urlPort = 80;
String urlPath = "/soccom/api/items/" + partNumber + "/reviews";
URI uri = URIUtils.createURI(
    urlScheme, urlHost, urlPort, urlPath,
    URLEncodedUtils.format(params, "UTF-8"), null);
if (LOGGER.isLoggable(Level.FINE)) {
    LOGGER.logp(Level.FINE, CLASS_NAME, METHOD_NAME,
        "URI: \"{0}\"", uri);
}
HttpGet method = new HttpGet(uri);

//TODO: make socket timeout configurable
method.getParams().setParameter("http.socket.timeout", 5000);

// Execute request
HttpResponse httpResponse = client.execute(method);
int statusCode = httpResponse.getStatusLine().getStatusCode();
if (LOGGER.isLoggable(Level.FINE)) {
    LOGGER.logp(Level.FINE, CLASS_NAME, METHOD_NAME,
        "HTTP status code: {0}", statusCode);
}

if (statusCode >= 200 && statusCode < 400) {
    // Read the response body.
    HttpEntity entity = httpResponse.getEntity();
    if (entity != null) {
        long len = entity.getContentLength();
        if (len != -1) {
            response = EntityUtils.toString(entity);
        }
    }

    if (LOGGER.isLoggable(Level.FINE)) {
        LOGGER.logp(Level.FINE, CLASS_NAME, METHOD_NAME,
            "HTTP response: \"{0}\"", response);
    }
}
else {
    String message = "Error code from soccom: "+statusCode;
    LOGGER.logp(Level.SEVERE, CLASS_NAME, METHOD_NAME,
        message);
    setErrorState(message, new Exception(message));
}

```



```

    }
    LOGGER.exiting(CLASS_NAME, METHOD_NAME, response);
    return response;
}

```

7. Add the `handleResponse` method as shown in Example 5-73 to the `ReviewsDataBean` class. This method uses the JSON API provided by `json.org` to parse the JSON object that is returned from the Social Commerce application.

The header-level return values are first retrieved and stored in attributes to the `ReviewsDataBean`. Then, each review returned is retrieved and a `ReviewDetailBean` is populated and added to the `reviewDetailsMap`, which is a mapping from review ID to `ReviewDetailBean` instances.

Example 5-73 The source code for the `handleResponse` method

```

private void handleResponse(String response) {
    final String METHOD_NAME = "handleResponse";
    LOGGER.entering(CLASS_NAME, METHOD_NAME, response);
    try {
        JSONObject json = new JSONObject(response);
        if (json.getInt("status") == 200) {
            JSONObject data = json.getJSONObject("data");

            JSONObject ratingStatistics = data
                .getJSONObject("ratingStatistics");
            numOfOverallRatings =
                ratingStatistics.getInt("numOfOverallRatings");
            if (numOfOverallRatings > 0) {
                avgOverallRating =
                    ratingStatistics.getDouble("avgOverallRating");
            }
            JSONObject ratingRange =
                ratingStatistics.getJSONObject("ratingRange");
            upperBound = ratingRange.getInt("upperBound");
            lowerBound = ratingRange.getInt("lowerBound");
            if (LOGGER.isLoggable(Level.FINE)) {
                LOGGER.logp(Level.FINE, CLASS_NAME, METHOD_NAME,
                    "numOfOverallRatings: {0}, avgOverallRating: {1},"
                    +" upperBound: {2}, lowerBound: {3}",
                    new Object[] {numOfOverallRatings,
                        avgOverallRating, upperBound, lowerBound} );
            }
            JSONObject paginationDetails =
                data.getJSONObject("paginationDetails");

```

```

sortDirection = paginationDetails
    .getString("sortDirection");
noOfEntriesPerPage = paginationDetails
    .getInt("noOfEntriesPerPage");
noOfTotalEntries = paginationDetails
    .getInt("noOfTotalEntries");
pageNum = paginationDetails.getInt("pageNum");
sortType = paginationDetails.getString("sortType");
noOfTotalPages =
    paginationDetails.getInt("noOfTotalPages");

JSONArray reviews = data.getJSONArray("reviews");
if (LOGGER.isLoggable(Level.FINE)) {
    LOGGER.logp(Level.FINE, CLASS_NAME, METHOD_NAME,
        "Found {0} reviews", reviews.length());
}
if (reviews.length() > 0) {
    JSONObject review = null;
    ReviewDetailBean reviewDetails = null;
    for (int i = 0; i < reviews.length(); i++) {
        review = reviews.getJSONObject(i);
        reviewDetails = new ReviewDetailBean();
        reviewDetails.setRatingOnly(review
            .getBoolean("isRatingOnly"));
        reviewDetails.setItemId(review.getString("itemId"));
        reviewDetails.setRating(review.getInt("rating"));
        if (LOGGER.isLoggable(Level.FINE)) {
            LOGGER.logp(Level.FINE, CLASS_NAME, METHOD_NAME,
                "rating ", review.getInt("rating"));
        }
        int reviewId = review.getInt("reviewId");
        reviewDetails.setReviewId(reviewId);
        reviewDetails.setReviewText(review
            .getString("reviewText"));
        reviewDetails.setSubmissionTime(review
            .getString("submissionTime"));
        reviewDetails.setTitle(review.getString("title"));
        reviewDetails.setUserId(review.getString("userId"));
        // Important! Use strings as the map key!
        reviewDetailsMap.put(String.valueOf(reviewId),
            reviewDetails);
        if (LOGGER.isLoggable(Level.FINE)) {
            LOGGER.logp(Level.FINE, CLASS_NAME, METHOD_NAME,
                "Review #{0}: {1}",
                new Object[] {i, reviewDetails} );
        }
    }
}

```

```

    }
    }
    }
} catch (Exception e) {
    LOGGER.logp(Level.SEVERE, CLASS_NAME, METHOD_NAME,
        "Exception in trying to parse review details", e);
    setErrorState("Exception in trying to parse review details",
        e);
}
LOGGER.exiting(CLASS_NAME, METHOD_NAME);
}

```

You will notice a number of errors due to missing import statements. You can remedy these errors using the Organize Imports function of IBM Rational Application Developer V7.5. However, given the number of classes that we depend on and the ambiguity of these classes, it is simpler to just add these manually.

8. Add the imports shown in Example 5-74 to the top of the `ReviewsDataBean` source code, just after the package statement. Save the file by pressing Ctrl+S.

Example 5-74 Import statements for ReviewsDataBean

```

import java.io.IOException;
import java.net.URI;
import java.net.URISyntaxException;
import java.util.LinkedHashMap;
import java.util.LinkedList;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;

import org.apache.http.HttpEntity;
import org.apache.http.HttpResponse;
import org.apache.http.NameValuePair;
import org.apache.http.client.ClientProtocolException;
import org.apache.http.client.HttpClient;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.client.utils.URIUtils;
import org.apache.http.client.utils.URLEncodedUtils;
import org.apache.http.impl.client.DefaultHttpClient;
import org.apache.http.message.BasicNameValuePair;
import org.apache.http.util.EntityUtils;
import org.json.JSONArray;

```

```
import org.json.JSONObject;

import com.ibm.commerce.beans.SmartDataBean;
import com.ibm.commerce.beans.SmartDataBeanImpl;
```

The ReviewsDataBean is now functionally complete, but to simplify debugging, you add a toString method that summarizes the information that the bean contains.

9. Add the toString method as shown in Example 5-75 to the ReviewsDataBean and press Ctrl+S to save the file.

Example 5-75 Source code for the toString method for ReviewsDataBean

```
public String toString() {
    StringBuffer buf = new StringBuffer();
    buf.append("{avgOverallRating=").append(avgOverallRating);
    buf.append(",lowerBound=").append(lowerBound);
    buf.append(",upperBound=").append(upperBound);
    buf.append(",noOfEntriesPerPage=").append(noOfEntriesPerPage);
    buf.append(",noOfTotalEntries=").append(noOfTotalEntries);
    buf.append(",noOfTotalPages=").append(noOfTotalPages);
    buf.append(",numOfOverallRatings=").append(numOfOverallRatings);
    buf.append(",onPage=").append(onPage);
    buf.append(",pageNum=").append(pageNum);
    buf.append(",pageSize=").append(pageSize);
    buf.append(",partNumber=").append(partNumber);
    buf.append(",sortDirection=").append(sortDirection);
    buf.append(",sortOptions=").append(sortOptions);
    buf.append(",sortType=").append(sortType);
    buf.append(",reviewDetailsMap=").append(reviewDetailsMap);
    buf.append(",errorMessage=").append(errorMessage);
    buf.append(",exception=").append(exception);
    buf.append("}");
    return buf.toString();
}
```

The ReviewsDataBean is now complete. You will notice some compilation errors relating to missing ReviewDetailBean. You create this bean in the next section and resolve the compilation errors.

Create the ReviewDetailBean class


The ReviewDetailBean class is a data transfer object (DTO) that encapsulates the Review detail data and has getter and setter for each of the values. This bean is used on JSP pages to display the review details.

To create the ReviewDetailBean class:

1. Create a new class, ReviewDetailBean, that inherits from java.lang.Object under the same package that is used for the ReviewsDataBean class.
2. Add the attributes to the class as shown in Example 5-76.

Example 5-76 Attributes for ReviewDetailBean

```
/** <code>>true</code> if this is a rating with no review */
private boolean isRatingOnly;
/** ID of this review. Used for later retrieving review details */
private int reviewId;
/** Main review text. Usually several paragraphs. */
private String reviewText;
/** ID of the submitter of the review */
private String userId;
/** A one-line title of the review */
private String title;
/** Timestamp of the submission in ISO format */
private String submissionTime;
/** Rating (0..5) */
private int rating;
/** The part number of the product being rated */
private String itemId;
```

3. Add accessor methods for these variables. To add the necessary accessor methods:
 - a. With the cursor within the class source code, select **Source**  **Generate Getters and Setters**.
 - b. When the Generate Getters and Setters window opens, click **Select All** and click **OK**.
 - c. Press Ctrl+S to save the file.
4. As shown in Example 5-68 on page 325, the submission time is returned in ISO format. For the JSPs that use the ReviewDetailBean to be able to format the submission time correctly for display, you need to introduce a method that parses the String version of the submission time stamp and returns it as a java.util.Date object. The JSPs can then use the date formatting tags within JSTL to format the time stamp. Add the method shown in Example 5-77 to provide this capability.

Example 5-77 Source code for getSubmissionTimeAsDate method

```
public java.util.Date getSubmissionTimeAsDate() {
    java.text.SimpleDateFormat sdf = new
        java.text.SimpleDateFormat("yyyy-MM-dd'T'hh:mm:ss.SSS");
```

```

    try {
        java.util.Date d = sdf.parse(getSubmissionTime());
        return d;
    } catch (java.text.ParseException e) {
        return null;
    }
}

```

5. As with the `ReviewsDataBean`, the final step is to add a `toString` method for debugging purposes. Add the method shown in Example 5-78, and press Ctrl+S to enable and save the file.

Example 5-78 Source code for `toString` method of `ReviewsDataBean`

```

public String toString() {
    StringBuffer buf = new StringBuffer();
    buf.append("{");
    buf.append("reviewId: ").append(reviewId);
    buf.append(", itemId: ").append(itemId);
    buf.append(", userId: ").append(userId);
    buf.append(", submissionTime: ").append(submissionTime);
    buf.append(", rating: ").append(rating);
    buf.append(", title: ").append(title);
    buf.append(", reviewText: ").append(reviewText);
    buf.append("}");
    return buf.toString();
}

```

5.5.5 Create the generic star display JSP

In recognition that all of the pages that show ratings and reviews need to be able to illustrate the current rating with a number of fully or partially filled out stars, you need to create a utility JSP that can be included on such pages. You create this JSP as a JSP that is included dynamically to ensure that the JSP is independent from the including JSP, as well as to enable the JSP to be configured for fragment caching.

Important: We do not consider caching configuration in this example. We recommend that you review the caching design of your site before using any of this code on a production environment. As an example, the Social Commerce application might return ratings with any number of fractional digits. We round the digits in this JSP, but a more complete approach from a caching perspective is to round in the data bean.

Because the actual scale used is part of the data that is returned from the Social Commerce server, the JSP needs to be capable of showing a rating using any scale. Also, you need to allow a fractional rating to display by showing partially filled stars, as shown in Figure 5-49.

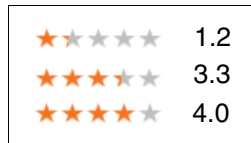


Figure 5-49 Three different star ratings

To create the star rating JSP, `StarRating.jsp`, follow these steps:

1. In the Java EE perspective of IBM WebSphere Commerce V7 Developer Edition, expand and right-click **Stores** \varnothing **WebContent** \varnothing **Madisons** \varnothing **mobile** \varnothing **Snippets** \varnothing **ReusableObjects** and select **New** \varnothing **File**.
2. In the New File wizard, enter `StarRating.jsp` in the File name entry field. Ensure that the parent folder entry field contains the following path, as shown in Figure 5-50, and click **Finish**:

`Stores/WebContent/Madisons/mobile/Snippets/ReusableObjects`

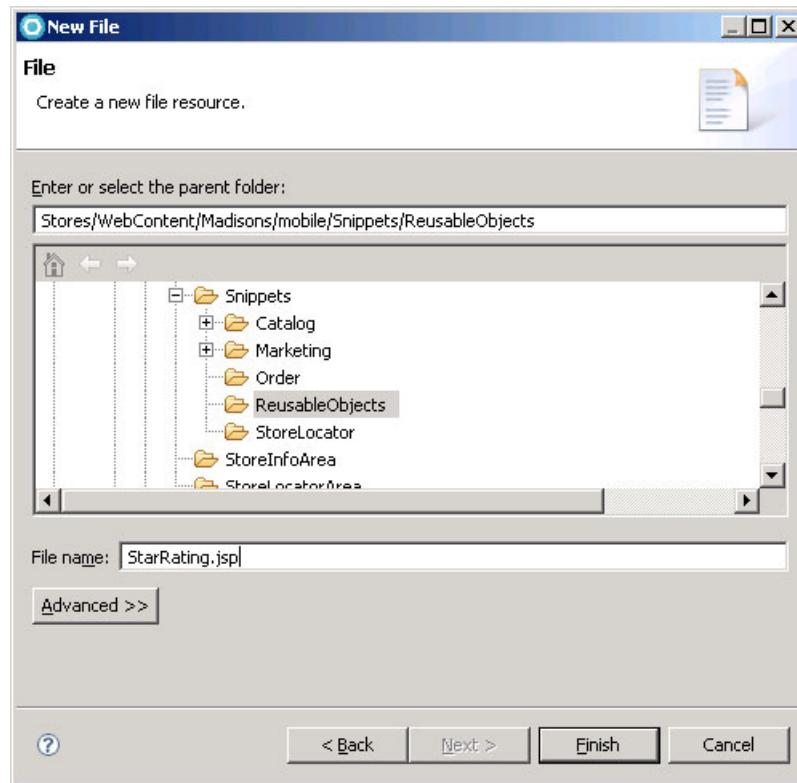


Figure 5-50 Entering the name for the new *StarRating.jsp* file in the new file wizard

3. The JSP editor opens. Add a comment to introduce the use of the utility JSP, declare the JSTL tags to be used, and define a few constants.
4. In the JSP editor, enter the code shown in Example 5-79. and press Ctrl+S to save the file.

*Example 5-79 Header and constants for *StarRating.jsp**

```
<%--
*****
* This JSP snippet displays a star rating
*
* Mandatory parameters:
*
* - lowerBound: The min value for a rating
* - upperBound: The max value for a rating
* - rating: The average rating (number of stars to display).
*           This does not have to be integer
*
```



```

* Sample invocation:
* <c:import url="${jspStoreDir}mobile/Snippets/ReusableObjects/StarRating.jsp">
*   <c:param name="lowerBound" value="0" /
*   <c:param name="upperBound" value="5" />
*   <c:param name="rating" value="3.7534" />
* </c:import>
*****
--%>

<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn" %>

<%-- Constants for image file names --%>
<c:set var="starImgPath" value="/soccom/ibm/social/images/" />
<c:set var="emptyStar" value="${starImgPath}star_grey.png" />
<c:set var="fullStar" value="${starImgPath}star_full.10.png" />
<c:set var="fractionStarPrefix" value="${starImgPath}star_full." />
<c:set var="fractionStarSuffix" value=".png" />

```

5. Add the code in Example 5-80 to the end of the `StarRating.jsp`, and press Ctrl+S to save the file.

Example 5-80 Retrieving parameters and determining the fractional rating for `StarRating.jsp`

```

<%-- Retrieve parameters --%>
<c:set var="lowerBound" value="${param.lowerBound}" />
<c:set var="upperBound" value="${param.upperBound}" />
<c:set var="rating" value="${param.rating}" />

<%-- determine integer and fraction parts --%>
<c:set var="ratingArr" value="${fn:split(rating, '.')}" />
<c:set var="ratingInt" value="${ratingArr[0]}" />
<c:set var="ratingFrac" value="0" />

<%-- was there a fractional part? --%>
<c:if test="${fn:length(ratingArr) > 1}">
  <%-- we only care about the first digit of the fractions (with rounding) --%>
  <c:set var="ratingFrac" value="${fn:substring(ratingArr[1],0,1)}" />
  <%-- round up if the second digit after the decimal point is above four --%>
  <c:if test="${fn:length(ratingArr[1])>1 && fn:substring(ratingArr[1],1,2)>4}">
    <c:set var="ratingFrac" value="${ratingFrac+1}" />
  </c:if>
</c:if>

<%-- define the accessibility text to be used for the images --%>
<c:set var="ratingAltText" value="${ratingInt}.${ratingFrac}" />

```

The code in Example 5-80 is very straightforward. First, it retrieves the parameters given to the JSP. Then, it uses string manipulation to isolate the integer and the fractional part of the rating, adding appropriate rounding if necessary. Finally, it defines the text to display if the images cannot be displayed (for example, if the page is rendered by a screen reader).

6. Complete the `StarRating.jsp` by adding the code in Example 5-81 to the end of the file. Press Ctrl+S to save the file.

Example 5-81 The logic to generate the stars for `StarRating.jsp`

```
<%-- First the full stars --%>
<c:forEach var="index" begin="${lowerBound+1}" end="${ratingInt}"><%--
    --%><%--
--%></c:forEach><%--

    Then the fraction part
--%><c:choose><%--
--%><c:when test="${ratingFrac > 0}"><%--
    --%><%--
        ...we "used" a star for the fraction, so display one less empty star...
    --%><c:set var="skip" value="2" /><%--
--%></c:when><%--
--%><c:otherwise><%--
    --%><c:set var="skip" value="1" /><%--
--%></c:otherwise><%--
--%></c:choose><%--

    And finally the empty stars
--%><c:forEach var="index" begin="${ratingInt+skip}" end="${upperBound}"><%--
    --%><%--
--%></c:forEach>
```

The code in Example 5-81 is the main logic, split into the following sections:

- ▶ The code to generate the full stars, given by the integer part of the rating
- ▶ The code to generate one optional fractional star
- ▶ The code to fill up with empty stars

Important: All HTML and JSTL tags in Example 5-81 are separated using JSP comments to ensure that all the `` tags appear in the resulting HTML document without any spacing. Even a single white space character ruins the layout of the stars.

The `StarRating.jsp` file is now complete.

5.5.6 Create the JSP fragment for the Write Review button

As mentioned earlier, you introduce a JSP fragment that contains the display logic for generating a button that links to the Write Review page if the current user is a registered shopper or links to the sign in and log on pages if the current user is a guest shopper.

To create this JSP snippet:

1. In the Package Explorer of IBM WebSphere Commerce V7 Developer Edition, expand and right-click **Stores** \oslash **WebContent** \oslash **Madisons** \oslash **mobile** \oslash **Snippets** \oslash **Catalog** and select **New** \oslash **Folder**.
2. The New Folder wizard opens. Enter Reviews in the Folder name field, and click **Finish**.
3. Right-click the newly created folder, and select **New** \oslash **File**.
4. The New File wizard opens. Enter WriteReviewLink.jspf in the File name field, and click **Finish**.
5. In the source code panel, enter the code from Example 5-82. This code defines the following URLs:

- writeReviewLink

This is the direct link to the write review page and is used when the JSP fragment is included on a page that is viewed by a registered shopper.

- logonURL

This is the link to the logon page. Notice that we specify the writeReviewLink value as the URL parameter value to utilize the fact that the mobile logon page allows the user to be redirected back to a certain page after logging on.

- registerURL

This is the link to the mobile registration page. Because this page does not support redirection back to a specific page, we specify only the common store parameters.

Example 5-82 URL generation code for the EnterWriteReviewLink.jspf file

```
<wcf:url var="writeReviewLink" value="mPostReviewsView">
  <c:forEach var="parameter" items="${WCParamValues}">
    <c:forEach var="value" items="${parameter.value}">
      <wcf:param name="${parameter.key}" value="${value}" />
    </c:forEach>
  </c:forEach>
  <wcf:param name="partNumber" value="${partNumber}" />
</wcf:url>
```

```

<wcf:url var="logonURL" value="MobileLogonForm">
  <wcf:param name="catalogId" value="${WCPParam.catalogId}" />
  <wcf:param name="storeId" value="${WCPParam.storeId}" />
  <wcf:param name="langId" value="${WCPParam.langId}" />
  <wcf:param name="URL" value="${writeReviewLink}" />
</wcf:url>
<wcf:url var="registerURL" value="MobileUserRegistrationAddForm">
  <wcf:param name="catalogId" value="${WCPParam.catalogId}" />
  <wcf:param name="storeId" value="${WCPParam.storeId}" />
  <wcf:param name="langId" value="${WCPParam.langId}" />
  <wcf:param name="register_button" value="Register" />
</wcf:url>

```

6. Add the code from Example 5-83 after the last line of code from Example 5-82.

This code implements the display logic. The code simply uses the `userType` attribute defined in `JSTLEnvironmentSetup.jspf` to determine whether the current user is a registered shopper. If the user is not a registered shopper, links for registration display. Otherwise, a form with a single button is displayed for writing a review.

Note: The form approach is mainly used to avoid any Web crawlers from accidentally generating reviews, although it is unlikely that such crawlers appear to the site as registered shoppers.

7. Save the file by pressing Ctrl+S.

Example 5-83 Display logic for the `EnterWriteReviewLink.jspf` file

```

<c:choose>
  <c:when test="${userType eq 'G'}">
    <!-- guest shopper -- show sign in or log on links -->
    <fmt:message key="WRITE_REVIEW_REGISTER_OR_LOGIN"
      bundle="${storeText}">
      <fmt:param value="${fn:escapeXml(registerURL)}" />
      <fmt:param value="${fn:escapeXml(logonURL)}" />
    </fmt:message>
  </c:when>
  <c:otherwise>
    <!-- registered user, show Write Review button -->
    <fmt:message var="writeReview" key="REVIEW_WRITE_REVIEW"
      bundle="${storeText}" />
    <form method="get" action="mPostReviewsView">
      <fieldset>

```

```

        <c:forEach var="parameter" items="${WCParmValues}">
            <c:forEach var="value" items="${parameter.value}">
                <input type="hidden" name="${parameter.key}"
                    value="${value}" />
            </c:forEach>
        </c:forEach>
        <input type="hidden" name="partNumber"
            value="${partNumber}" />
        <input type="submit" name="review"
            value="${writeReview}" />
    </fieldset>
</form>
</c:otherwise>
</c:choose>

```

The Write Review button JSP fragment is now complete and can be included statically on all pages that need such a button.

5.5.7 Create the JSP fragment for the review header

Another JSP fragment that is needed on several pages is the review header, as shown in Figure 5-51. The JSP fragment must be able to display the “Read review” link conditionally.

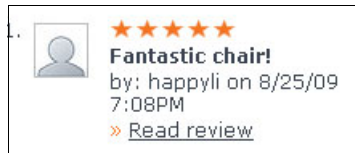


Figure 5-51 Review header

To create this JSP fragment:

1. In the Package Explorer of IBM WebSphere Commerce V7 Developer Edition, expand and right-click **Stores** \emptyset **WebContent** \emptyset **Madisons** \emptyset **mobile** \emptyset **Snippets** \emptyset **Catalog** \emptyset **Reviews** and select **New** \emptyset **File**.
2. The New File wizard opens. Enter `ReviewHeadingDisplay.jspf` in the File name field, and click **Finish**.
3. In the source code panel, enter the code from Example 5-84, which defines three URLs.

Example 5-84 Full source code for the ReviewHeadingDisplay.jspf file

```
<%--
```

This code can be statically included by any page that has instantiated an object of type `ReviewsDataBean` with the attribute `ID reviewDetails`.

```
--%>
<a href="#" class="user_image">
  
</a>
<ul>
  <li>
    <%out.flush();%>
    <c:import
url="${jspStoreDir}mobile/Snippets/ReusableObjects/StarRating.jsp">
      <c:param name="lowerBound" value="${reviews.lowerBound}" />
      <c:param name="upperBound" value="${reviews.upperBound}" />
      <c:param name="rating" value="${reviewDetails.rating}" />
    </c:import>
    <%out.flush();%>
  </li>
  <li><h4><c:out value="${reviewDetails.title}" /></h4></li>
  <li>
    <!-- retrieve date format and format the review date --%>
    <fmt:message var="dateFormat" key="REVIEW_DATEFORMAT"
      bundle="${storeText}" />
    <fmt:message key="REVIEW_OVERVIEW_INFO" bundle="${storeText}">
      <fmt:param value="${reviewDetails.userId}" />
      <fmt:param><fmt:formatDate pattern="${dateFormat}"
        value="${reviewDetails.submissionTimeAsDate}"
      /></fmt:param>
    </fmt:message>
  </li>
  <c:if test="${!empty readReviewLink}">
    <fmt:message var="readReview" key="REVIEW_READ_REVIEW"
      bundle="${storeText}" />
    <li>
      <span class="bullet">&#187; </span>
      <a href="${fn:escapeXml(readReviewLink)}' />"
        title="${fn:escapeXml(readReview)}">
        ${readReview}
      </a>
    </li>
  </c:if>
</ul>
```

Important: In the review list, we display a static avatar image (/soccom/small.jpg) for the user profile picture. You can choose to display a dynamic picture that is retrieved from a social profile for the user.

4. Save the file by pressing Ctrl+S.

The ReviewHeadingDisplay.jspf JSP snippet is now complete. As the JSP comment in the top of Example 5-84 states, the snippet can be used as a static include by an including page that has already instantiated a ReviewsDataBean using the attribute reviewDetails.

5.5.8 Modify the product display pages

As illustrated in Figure 5-42 on page 314, you need to amend the product page with the ability to show the overall rating for a product along with a link to the review details for that product. Because the product display page is in fact divided across four separate display pages, you need to create a JSP fragment that can be included from any of those pages to minimize code duplication across the four product display pages. We call this page RatingDisplay.jspf and place it in the WebContent\Madisons\mobile\Snippets\Catalog\CatalogEntryDisplay folder within the Stores project.

In this section, we explain how to modify the product display pages in the following topics:

- ▶ Create the RatingDisplay.jspf fragment
- ▶ Modify the CachedProductDisplay.jsp file
- ▶ Modify the CachedItemDisplay.jsp file
- ▶ Modify the CachedBundleDisplay.jsp file
- ▶ Modify the CachedPackageDisplay.jsp file

In this example, we use the part number for the product ID that is displayed, which results in different review entries for a product and each of its SKUs. In your implementation, you might want to collate all the reviews for a number of SKUs under the parent product.

Create the RatingDisplay.jspf fragment

To create this fragment:

1. In the Java EE perspective of IBM WebSphere Commerce V7 Developer Edition, expand and right-click **Stores** ▾ **WebContent** ▾ **Madisons** ▾ **mobile** ▾ **Snippets** ▾ **Catalog** ▾ **Reviews** and select **New** ▾ **File**.
2. In the New File wizard, enter `RatingDisplay.jspf` in the File name entry field. Ensure that the parent folder entry field contains the following path:
`Stores/WebContent/Madisons/mobile/Snippets/Reviews/CatalogEntryDisplay`
Click **Finish**.
3. The JSP editor opens. Enter the code from Example 5-85, and then press `Ctrl+S` to save the file.

Example 5-85 Code for RatingDisplay.jspf to retrieve the ratings details

```
<%-- BEGIN overall rating code --%>

<wcbase:useBean id="reviewCatentry"
    classname="com.ibm.commerce.catalog.beans.CatalogEntryDataBean"
    scope="request" />

<c:set var="reviewItem" value="${reviewCatentry.partNumber}" />

<%-- instantiate the review smart data bean to retrieve the review details --%>
<wcbase:useBean id="reviews"
    classname="com.ibm.itso.commerce.soccom.beans.ReviewsDataBean"
    scope="request">
    <c:set property="pageSize" value="1" target="${reviews}" />
    <c:set property="onPage" value="1" target="${reviews}" />
    <c:set property="sortOptions" value="rating|desc" target="${reviews}" />
    <c:set property="partNumber" value="${reviewItem}" target="${reviews}" />
</wcbase:useBean>
```

The code in Example 5-85 retrieves the review information for the current product, item, bundle, or package. It instantiates a `CatalogEntryDataBean` using the request information in order to retrieve the part number of the current catalog entry. Then, an instance of the `ReviewsDataBean` class is created and populated using the parameters relevant to this page.

Because we are interested only in overview information and because the overview information is the same regardless of the number of reviews that we retrieve, we ask to retrieve only a single review by specifying a page size of 1.

4. Add the code in Example 5-86 at the end of `RatingsDisplay.jspf`, and press Ctrl+S to save the file.

Example 5-86 Link information initialization code

```
<%-- prepare the link text for the review details --%>
<fmt:message var="readReviewsLinkTitle"
    key="RATING_READ_REVIEWS_LINK" bundle="${storeText}" />

<%-- prepare the link target for the review details --%>
<wcf:url var="readReviewsLink" value="mProductReviewsView">
    <c:forEach var="parameter" items="${WCPParamValues}">
        <c:forEach var="value" items="${parameter.value}">
            <wcf:param name="${parameter.key}" value="${value}" />
        </c:forEach>
    </c:forEach>
    <wcf:param name="pgGrp" value="${WCPParam.pgGrp}" />
    <wcf:param name="onPage" value="1" />
    <wcf:param name="sortOptions" value="rating|desc"/>
</wcf:url>

<%-- prepare the link text for writing a review --%>
<fmt:message var="writeReviewLinkTitle"
    key="RATING_WRITE_FIRST_REVIEW_LINK" bundle="${storeText}" />
```

The code in Example 5-86 prepares the text and URL for reading the reviews. Now that everything is initialized, you can add the main logic to `RatingDisplay.jspf`.

5. Add the code in Example 5-87 to the end of `RatingDisplay.jspf`, and press Ctrl+S to save the file.

Example 5-87 Main logic for `RatingDisplay.jspf`

```
<ul id="product_rating">
    <c:choose>
        <%-- an error occurred retrieving the review information --%>
        <c:when test="${reviews.error}">
            <li><fmt:message key="RATING_OVERALL_ERROR" bundle="${storeText}" /></li>
            <li><span class="bullet">&#187; </span>
                <a href="${readReviewsLink}"
                    title="<c:out value='${readReviewsLinkTitle}' />">
                    <c:out value="${readReviewsLinkTitle}" />
                </a>
            </li>
        </c:when>
        <%-- no reviews yet --%>
```

```

<c:when test="${reviews.numOfOverallRatings eq 0}">
  <li><fmt:message key="RATING_OVERALL_NONE" bundle="${storeText}" /></li>
  <li><%@include
    file="../../Snippets/Catalog/Reviews/WriteReviewLink.jspf" %></li>
</c:when>
<!-- one or more reviews -->
<c:otherwise>
  <li><fmt:message key="RATING_OVERALL_TITLE" bundle="${storeText}" />
  <%out.flush();%>
  <c:import
    url="${jspStoreDir}mobile/Snippets/ReusableObjects/StarRating.jsp">
    <c:param name="lowerBound" value="${reviews.lowerBound}" />
    <c:param name="upperBound" value="${reviews.upperBound}" />
    <c:param name="avgOverallRating" value="${reviews.avgOverallRating}" />
  </c:import>
  <%out.flush();%>
</li>
  <li><%@include
    file="../../Snippets/Catalog/Reviews/WriteReviewLink.jspf" %></li>
</c:otherwise>
</c:choose>
</ul>
<!-- END overall rating code -->

```

The code in Example 5-87 creates one of the following occurrences:

- An error message if an error occurred while attempting to retrieve the ratings
- A link to the Write a Review page if no ratings were found for the product
- The overall average rating with a link to the review details if one or more reviews were found

The RatingDisplay.jspf fragment is now complete.

Modify the CachedProductDisplay.jsp file

To add a rating overview to the CachedProductDisplay.jsp file:

1. In the Java EE perspective of IBM WebSphere Commerce V7 Developer Edition, expand and double-click **Stores**  **WebContent**  **Madisons**  **mobile**  **Snippets**  **Catalog**  **CatalogEntryDisplay**  **CachedProductDisplay.jsp**.
2. The CachedProductDisplay.jsp file opens in the JSP editor. If the source tab is not active, then switch to it by clicking **Source** in the bottom-left of the panel.

3. Press Ctrl+F to open the Find/Replace window, enter AddToProdCompare in the Find entry field, and click **Find**.

The code snippet shown in Example 5-88 is displayed.

Example 5-88 Code snippet from CachedProductDisplay.jsp

```
<p><span class="bullet">&#187; </span> <a href="#"
onclick="javascript:resolveSKU('<c:out value="{itemsAttributes}" />', '<c:out
value="{selValSeparator}" />');" title="{fn:escapeXml(showStoreAvail)}"><c:out
value="{showStoreAvail}" /></a></p>
</c:if>

<wcf:url var="AddToProdCompare" value="mAddToProdCompare">
  <c:forEach var="parameter" items="{WCParamValues}">
    <c:forEach var="value" items="{parameter.value}">
```

4. Insert the following line between the </c:if> and the <wcf:url> lines in Figure 5-88. Then, press Ctrl+S to save the file.

```
<%@ include file="../Reviews/RatingDisplay.jspf" %>
```

The resulting code should be similar to the code shown in Example 5-89 with the inserted code highlighted in bold font.

Example 5-89 Code snippet from CachedProductDisplay.jsp with rating added







```
<p><span class="bullet">&#187; </span> <a href="#"
onclick="javascript:resolveSKU('<c:out value="{itemsAttributes}" />', '<c:out
value="{selValSeparator}" />');" title="{fn:escapeXml(showStoreAvail)}"><c:out
value="{showStoreAvail}" /></a></p>
</c:if>

<%@ include file="../Reviews/RatingDisplay.jspf" %>

<wcf:url var="AddToProdCompare" value="mAddToProdCompare">
  <c:forEach var="parameter" items="{WCParamValues}">
    <c:forEach var="value" items="{parameter.value}">
```

Modify the CachedItemDisplay.jsp file

To add the rating overview to the CachedItemDisplay.jsp file:

1. In the Java EE perspective of IBM WebSphere Commerce V7 Developer Edition, expand and double-click **Stores**  **WebContent**  **Madisons**  **mobile**  **Snippets**  **Catalog**  **CatalogEntryDisplay**  **CachedItemDisplay.jsp**.
2. The CachedItemDisplay.jsp file opens in the JSP editor. If the source tab is not active, then switch to it by clicking **Source** in the bottom-left of the panel.
3. Press Ctrl+F to open the Find/Replace window, enter WishlistDisplayURL in the Find entry field, and click **Find**.

The code snippet shown in Example 5-90 is displayed.

Example 5-90 Code snippet from CachedItemDisplay.jsp

```
</c:import>
</li>

<c:if test="${isBuyable}">
  <c:set var="WishlistDisplayURL"
    value="InterestItemDisplay?URL=mInterestListDisplay&listId=" />
  <wcf:url var="AddToWishlist" value="InterestItemAdd">
    <wcf:param name="catEntryId" value="${item.itemID}"/>
  </wcf:url>
</c:if>
```

4. Insert the following line between the and the <c:if> lines in Example 5-90, and press Ctrl+S to save the file:

```
<%@ include file="../Reviews/RatingDisplay.jspf" %>
```

The resulting code should be similar to the code shown in Example 5-91 with the inserted code highlighted in bold font.

Example 5-91 Code snippet from CachedItemDisplay.jsp with rating added








```
</c:import>
</li>

<%@ include file="../Reviews/RatingDisplay.jspf" %>

<c:if test="${isBuyable}">
  <c:set var="WishlistDisplayURL"
    value="InterestItemDisplay?URL=mInterestListDisplay&listId=" />
  <wcf:url var="AddToWishlist" value="InterestItemAdd">
    <wcf:param name="catEntryId" value="${item.itemID}"/>
  </wcf:url>
</c:if>
```

Modify the CachedBundleDisplay.jsp file

To add the rating overview to the CachedBundleDisplay.jsp file:

1. In the Java EE perspective of IBM WebSphere Commerce V7 Developer Edition, expand and double-click **Stores**  **WebContent**  **Madisons**  **mobile**  **Snippets**  **Catalog**  **CatalogEntryDisplay**  **CachedBundleDisplay.jsp**.
2. The CachedBundleDisplay.jsp file opens in the JSP editor. If the source tab is not active, then switch to it by clicking **Source** in the bottom-left of the panel.
3. Press Ctrl+F to open the Find/Replace window, enter submitAddToCart() in the Find entry field, and click **Find**.

The code snippet shown in Example 5-92 is displayed.

Example 5-92 Code snippet from CachedBundleDisplay.jsp

```
<li>
  <input type="button" onclick="javascript:submitAddToCart();" name="add_to_cart"
id="add_to_cart" value="<fmt:message key="ADD_TO_CART" bundle="${storeText}" />" />
</li>
<li>
  <span class="bullet">&#187; </span>
```

4. Insert the following line between the and the lines in Example 5-92, and press Ctrl+S to save the file:

```
<%@ include file="../Reviews/RatingDisplay.jspf" %>
```

The resulting code should be similar to the code shown in Example 5-93 with the inserted code highlighted in bold.

Example 5-93 Code snippet from CachedBundleDisplay.jsp with rating added

```
<li>
  <input type="button" onclick="javascript:submitAddToCart();" name="add_to_cart"
id="add_to_cart" value="<fmt:message key="ADD_TO_CART" bundle="${storeText}" />" />
</li>
<%@ include file="../Reviews/RatingDisplay.jspf" %>
<li>
  <span class="bullet">&#187; </span>
```

Modify the CachedPackageDisplay.jsp file

To add the rating overview to the CachedPackageDisplay.jsp file:

1. In the Java EE perspective of IBM WebSphere Commerce V7 Developer Edition, expand and double-click **Stores** **WebContent** **Madisons** **mobile** **Snippets** **Catalog** **CatalogEntryDisplay** **CachedPackageDisplay.jsp**.
2. The CachedPackageDisplay.jsp file opens in the JSP editor. If the source tab is not active, then switch to it by clicking **Source** in the bottom-left of the panel.
3. Press Ctrl+F to open the Find/Replace window, enter submitAddToCart() in the Find entry field, and click **Find**.

The code snippet shown in Example 5-94 is displayed.

Example 5-94 Code snippet from CachedPackageDisplay.jsp

```
<input type="button" onclick="javascript:submitAddToCart();" name="add_to_cart"
id="add_to_cart" value="<fmt:message key="ADD_TO_CART" bundle="{storeText}" />" />
</li>
    <span class="bullet">&#187; </span>
    <a href="#" onclick="document.getElementById('wishlist_add').submit();"
title="<fmt:message key="WISHLIST" bundle="{storeText}" />"><fmt:message
key="WISHLIST" bundle="{storeText}" /></a>
</li>
```

4. Insert the following line between the <input> and the lines in Example 5-94, and press Ctrl+S to save the file.

```
<%@ include file="../Reviews/RatingDisplay.jspf" %>
```

The resulting code should be similar to the code shown in Example 5-95 with the inserted code highlighted in bold font.

Example 5-95 Code snippet from CachedPackageDisplay.jsp with rating added

```
<input type="button" onclick="javascript:submitAddToCart();" name="add_to_cart"
id="add_to_cart" value="<fmt:message key="ADD_TO_CART" bundle="{storeText}" />" />








<%@ include file="../Reviews/RatingDisplay.jspf" %>

</li>
    <span class="bullet">&#187; </span>
    <a href="#" onclick="document.getElementById('wishlist_add').submit();"
title="<fmt:message key="WISHLIST" bundle="{storeText}" />"><fmt:message
key="WISHLIST" bundle="{storeText}" /></a>
</li>
```

5.5.9 Modify the product compare page

As mentioned earlier, one of the major factors behind a shopper's buying decision on the product compare page is the relative product ratings of the products that are compared. You add the overall rating to the product compare page between the price and the inventory availability sections, as shown in Figure 5-43 on page 315. The product compare page is generated by the `ProductCompareResultGridDisplay.jsp` file, so you need to modify this JSP to add the ratings.

To amend the product compare page with overall product ratings:

1. In the Java EE perspective of IBM WebSphere Commerce V7 Developer Edition, expand and double-click **Stores**  **WebContent**  **Madisons**  **mobile**  **ShoppingArea**  **CatalogSection**  **CatalogEntrySubsection**  **ProductCompareResultGridDisplay.jsp**.
2. The `ProductCompareResultGridDisplay.jsp` file opens in the JSP editor. If the source tab is not active, then switch to it by clicking **Source** in the bottom-left of the panel.
3. Press Ctrl+F to open the Find/Replace window, enter `online_availability` in the Find entry field, and click **Find**.
4. The code shown in Example 5-96 is displayed. This example is the part of the code between the price display and the availability display on the product compare page.

Example 5-96 Division between price display and the availability display in product compare

```
<div class="clear_float"></div>
</div>
</div>

<div id="online_availability">
  <div class="grid_row compare_criteria">
```

5. Add the code in Example 5-97 before the highlighted line in Example 5-96, for example between the end of the previous `<div>` tag and the beginning of the `<div>` tag with the ID of `online_availability`.

Example 5-97 Overall rating display for product compare

```
<%-- BEGIN: Rating and Reviews --%>
<fmt:message var="readReviewsLinkTitle" key="RATING_READ_REVIEWS_LINK"
bundle="${storeText}" />
<div id="overall_review">
  <div class="grid_row compare_criteria">
    <div class="grid_column">
```

```


354 Building Multichannel Applications with WebSphere Commerce


```



```

</c:when>
<c:otherwise>
  <a href="${fn:escapeXml(readReviewsLink)}"
    title="${fn:escapeXml(readReviewsLinkTitle)}">
    <fmt:message key="PROD_CMPR_RATING_MULTIPLE"
      bundle="${storeText}">
      <fmt:param value="${reviews.numOfOverallRatings}" />
    </fmt:message>
  </a>
</c:otherwise>
</c:choose>
</div>
</c:forEach>
<div class="clear_float"></div>
</div>
<!-- END: Rating and Reviews -->

```








6. Save and close the file.

The code in Example 5-97 creates a new row in the product compare grid with columns for each of the products in the product comparison. Each product shows either a static text of “no reviews” or the rating with a link to the reviews. There are two special cases that support different localization strings in the case of only one review or more than one review.

5.5.10 Create the JSP for the review list and overview

You create the `ProductReviewList.jsp` file to display the list of reviews for a product and to enable the user to sort products based on the rating or the submission time and to paginate between the multiple review pages for a product.

To create the review list JSP, `ProductReviewList.jsp`:

1. In the Package Explorer of IBM WebSphere Commerce V7 Developer Edition, expand and right-click **Stores**  **WebContent**  **Madisons**  **mobile**  **ShoppingArea**  **CatalogSection** and select **New**  **Folder**.
2. The New Folder wizard opens. Enter `ReviewSubsection` in the Folder name field, and click **Finish**.
3. Right-click the newly created folder, and select **New**  **File**.
4. The New File wizard opens. Enter `ProductReviewList.jsp` in the File name field, and click **Finish**.

5. Next, add a comment to introduce the use of JSP, declare the JSTL tags to be used and define a few constants. In the JSP editor, enter the skeleton HTML document code shown in Example 5-98, and press Ctrl+S to save the file.

We expand sections of this code in the following steps.

Example 5-98 Skeleton code for ProductReviewList.jsp

```
<%--
    *****
    * This JSP displays the product review list.
    *****
--%>

<!-- BEGIN ProductReviewList.jsp -->

<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="http://commerce.ibm.com/base" prefix="wcbase" %>
<%@ taglib uri="http://commerce.ibm.com/foundation" prefix="wcf" %>

<%@ include file="../../../include/parameters.jspf" %>
<%@ include file="../../../include/JSTLEnvironmentSetup.jspf" %>

<%-- 1: Initialize beans and URL links --%>

<!DOCTYPE html PUBLIC "-//WAPFORUM//DTD XHTML Mobile 1.2//EN"
"http://www.openmobilealliance.org/tech/DTD/xhtml-mobile12.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" lang="${shortLocale}"
xml:lang="${shortLocale}">
  <head>
    <title>
      <fmt:message key="REVIEW_LIST_TITLE" bundle="${storeText}" />:
      <c:out value="${catalogEntry.description.name}" escapeXml="false"/>
    </title>
    <meta http-equiv="content-type" content="application/xhtml+xml" />
    <meta http-equiv="cache-control" content="max-age=300" />
    <meta name="viewport"
      content="width=device-width, initial-scale=1.0, user-scalable=no" />
    <link type="text/css" rel="stylesheet" href="${cssPath}"/>
  </head>

  <body>
    <div id="wrapper">
```

```

<%@ include file="../../../include/HeaderDisplay.jspf" %>
<%@ include file="../../../include/BreadCrumbTrailDisplay.jspf"%>

<%out.flush();%>

<div id="reviews" class="content_box">
  <div class="heading_container">
    <h2><fmt:message key="REVIEW_LIST_TITLE" bundle="${storeText}" /></h2>
    <div class="clear_float"></div>
  </div>

  <%-- 2. Main content --%>

</div>

<%out.flush();%>
<%@ include file="../../../include/FooterDisplay.jspf" %>
</div>
</body>
</html>
<!-- END ProductReviewList.jsp -->

```

6. Now, add the initialization code shown in Example 5-99 after the following line:

```
<%-- 1: Initialize beans and URL links --%>
```

The code first initializes variables used for the breadcrumb display JSP fragment and instantiates a `CatalogEntryDataBean` to retrieve the part number for the product in question. It then instantiates and populates an instance of the `ReviewsDataBean` to retrieve the reviews for that product.

The `pageSize` property can be loaded from a properties file, but in our example, we hard-coded the value to five results per page.

The `onPage` property is used for current page number, and the `sortOptions` property is used for specifying the sorting field and direction. These values are available in the request for the JSP.

Finally, a link to the product display page is generated for use later on in the page.

Example 5-99 Initialization code for ProductReviewList.jsp

```

<c:set var="pgGrp" value="${WCPParam.pgGrp}" />
<c:choose>
  <c:when test='${pgGrp == "prodComp"}'>

```

```

        <c:set var="prodComparePageGroup" value="true" scope="request"/>
    </c:when>
    <c:when test='${pgGrp == "search"}'>
        <c:set var="searchPageGroup" value="true" scope="request"/>
    </c:when>
    <c:when test='${pgGrp == "wishlist"}'>
        <c:set var="wishlistPageGroup" value="true" scope="request"/>
    </c:when>
    <c:otherwise>
        <c:set var="categoryNavPageGroup" value="true" scope="request"/>
    </c:otherwise>
</c:choose>

<c:set var="productReviewPage" value="true" scope="request"/>
<c:set var="productReviewListPage" value="true" scope="request"/>

<wcbase:useBean id="catalogEntry"
    classname="com.ibm.commerce.catalog.beans.CatalogEntryDataBean" />
<c:set var="partNumber" value="${catalogEntry.partNumber}" />

<wcbase:useBean id="reviews"
    classname="com.ibm.itso.commerce.socom.beans.ReviewsDataBean">
    <c:set property="pageSize" value="3" target="${reviews}"/>
    <c:set property="onPage" value="${WCPParam.onPage}" target="${reviews}"/>
    <c:set property="sortOptions" value="${WCPParam.sortOptions}" target="${reviews}"/>
    <c:set property="partNumber" value="${partNumber}" target="${reviews}"/>
</wcbase:useBean>

<wcf:url var="productDisplayLink" value="mProductDisplayView">
    <c:forEach var="parameter" items="${WCPParamValues}">
        <c:forEach var="value" items="${parameter.value}">
            <c:if test='${parameter.key != "reviewId"}'>
                <wcf:param name="${parameter.key}" value="${value}" />
            </c:if>
        </c:forEach>
    </c:forEach>
</wcf:url>

```

7. Add the code shown in Example 5-100 after the following line:

```
<%-- 2. Main content --%>
```

This code implements the following logic to display the overall star rating and Write Review link:

- a. Checks whether the data bean returns error. If there is a error, displays the message. This message can be picked from the properties file that was created in “Define localized strings for pages” on page 397.
- b. Writes the code to display the product name if there is no error.
- c. Imports the `StarRating.jsp` file to display the star rating for the product. Passes the `lowerBound`, `upperBound`, and `avgOverallRating` parameters to this JSP. The value for these parameters is retrieved from the `ReviewsDataBean` that was instantiated in step 6 on page 357.
- d. Checks whether the user is logged in or is a guest user. If the user is logged in, adds a link to Write Review. If not, then presents the user a message to Register or SignIn to write a review.

Example 5-100 Overall star rating and Write Review link

```
<c:choose>
  <c:when test="${reviews.error}">
    <p><fmt:message key="RATING_OVERALL_ERROR" bundle="${storeText}" /></p>
  </c:when>

  <c:otherwise>
    <p>
      <a href="${fn:escapeXml(productDisplayLink)}">
        <c:out value="${catalogEntry.description.name}" />
      </a>
    </p>
    <p><fmt:message key="RATING_OVERALL_TITLE" bundle="${storeText}" />
    <%out.flush();%>
    <c:import url="${jspStoreDir}mobile/Snippets/ReusableObjects/StarRating.jsp">
      <c:param name="lowerBound" value="${reviews.lowerBound}" />
      <c:param name="upperBound" value="${reviews.upperBound}" />
      <c:param name="rating" value="${reviews.avgOverallRating}" />
    </c:import>
    <%out.flush();%>
  </p>
  <%@include file="../../Snippets/Catalog/Reviews/WriteReviewLink.jspf" %>
```

8. Implement the sorting control as given in Example 5-101. Add this code after the code from Example 5-100 on page 359.

Example 5-101 Sorting control

```
<div class="sort_by_control">
  <fmt:message key="REVIEW_LIST_SORT_BY" bundle="${storeText}" />
  <c:choose>
    <c:when test="${empty WParam.sortOptions ||
      fn:startsWith(WParam.sortOptions, 'rating')}">
      <wcf:url var="ProductReviewListSortURL" value="mProductReviewsView">
        <c:forEach var="parameter" items="${WParamValues}">
          <c:if test="${parameter.key != 'sortOptions'}">
            <c:forEach var="value" items="${parameter.value}">
              <wcf:param name="${parameter.key}" value="${value}" />
            </c:forEach>
          </c:if>
        </c:forEach>
        <wcf:param name="sortOptions" value="submissionTime|desc" />
      </wcf:url>
      <span class="current_sort">
        <fmt:message key="REVIEW_LIST_SORT_BY_RATING" bundle="${storeText}" />
      </span>
      <a href="${fn:escapeXml(ProductReviewListSortURL)}">
        <fmt:message key="REVIEW_LIST_SORT_BY_NEWEST" bundle="${storeText}" />
      </a>
    </c:when>
    <c:otherwise>
      <wcf:url var="ProductReviewListSortURL" value="mProductReviewsView">
        <c:forEach var="parameter" items="${WParamValues}">
          <c:if test="${parameter.key != 'sortOptions'}">
            <c:forEach var="value" items="${parameter.value}">
              <wcf:param name="${parameter.key}" value="${value}" />
            </c:forEach>
          </c:if>
        </c:forEach>
        <wcf:param name="sortOptions" value="rating|desc" />
      </wcf:url>
      <a href="${fn:escapeXml(ProductReviewListSortURL)}">
        <fmt:message key="REVIEW_LIST_SORT_BY_RATING" bundle="${storeText}" />
      </a>
      <span class="current_sort">
        <fmt:message key="REVIEW_LIST_SORT_BY_NEWEST" bundle="${storeText}" />
      </span>
    </c:otherwise>
  </c:choose>
</div>
```

9. Write the code to display the list of reviews. Implement this code as shown in Example 5-102. Add the code after the code from Example 5-101 on page 360.

Example 5-102 List of reviews

```
<ol class="list_reviews">
  <c:forEach var="reviewDet" items="${reviews.reviewDetailsMap}" varStatus="status">
    <c:set var="reviewDetails" value="${reviewDet.value}" />
    <wcf:url var="readReviewLink" value="mReviewDetails">
      <c:forEach var="parameter" items="${WCPParamValues}">
        <c:forEach var="value" items="${parameter.value}">
          <wcf:param name="${parameter.key}" value="${value}" />
        </c:forEach>
      </c:forEach>
      <wcf:param name="reviewId" value="${reviewDet.value.reviewId}"/>
    </wcf:url>
    <li>
      <%@include
        file="../../Snippets/Catalog/Reviews/ReviewHeadingDisplay.jspf" %>
    </li>
  </c:forEach>
</ol>
```

10. Implement the Pagination Control as given in Example 5-103. Add this code after the code from Example 5-102 on page 361.

Example 5-103 Pagination control code

```
<div class="paging_control">
  <wcf:url var="ProductReviewListPrevURL" value="mProductReviewsView">
    <wcf:param name="langId" value="${langId}" />
    <wcf:param name="storeId" value="${WCPParam.storeId}" />
    <wcf:param name="catalogId" value="${WCPParam.catalogId}" />
    <wcf:param name="productId" value="${WCPParam.productId}" />
    <wcf:param name="onPage" value="${reviews.pageNum - 1}" />
    <wcf:param name="sortOptions" value="${WCPParam.sortOptions}"/>
    <wcf:param name="pgGrp" value="${WCPParam.pgGrp}" />

    <c:if test="${!empty WCPParam.pgGrp}">
      <c:choose>
        <c:when test="${WCPParam.pgGrp == 'catNav'}">
          <wcf:param name="categoryId" value="${WCPParam.categoryId}" />
          <wcf:param name="parent_category_rn"
value="${WCPParam.parent_category_rn}" />
          <wcf:param name="top_category" value="${WCPParam.top_category}" />
        </c:when>
      </c:choose>
    </c:if>
  </wcf:url>
</div>
```

```

        <wcf:param name="sequence" value="{WCParm.sequence}" />
    </c:when>
    <c:when test="{WCParm.pgGrp == 'search'}">
        <wcf:param name="resultCatEntryType"
value="{WCParm.resultCatEntryType}" />
        <wcf:param name="pageSize" value="{WCParm.pageSize}" />
        <wcf:param name="searchTerm" value="{WCParm.searchTerm}" />
        <wcf:param name="beginIndex" value="{WCParm.beginIndex}" />
        <wcf:param name="sType" value="{WCParm.sType}" />
    </c:when>
</c:choose>
</c:if>
</wcf:url>
<wcf:url var="ProductReviewListNextURL" value="mProductReviewsView">
    <wcf:param name="langId" value="{langId}" />
    <wcf:param name="storeId" value="{WCParm.storeId}" />
    <wcf:param name="catalogId" value="{WCParm.catalogId}" />
    <wcf:param name="productId" value="{WCParm.productId}" />
    <wcf:param name="onPage" value="{reviews.pageNum + 1}" />
    <wcf:param name="sortOptions" value="{WCParm.sortOptions}" />
    <wcf:param name="pgGrp" value="{WCParm.pgGrp}" />

    <c:if test="{!empty WCParm.pgGrp}">
        <c:choose>
            <c:when test="{WCParm.pgGrp == 'catNav'}">
                <wcf:param name="categoryId" value="{WCParm.categoryId}" />
                <wcf:param name="parent_category_rn"
value="{WCParm.parent_category_rn}" />
                <wcf:param name="top_category" value="{WCParm.top_category}" />
                <wcf:param name="sequence" value="{WCParm.sequence}" />
            </c:when>
            <c:when test="{WCParm.pgGrp == 'search'}">
                <wcf:param name="resultCatEntryType"
value="{WCParm.resultCatEntryType}" />
                <wcf:param name="pageSize" value="{WCParm.pageSize}" />
                <wcf:param name="searchTerm" value="{WCParm.searchTerm}" />
                <wcf:param name="beginIndex" value="{WCParm.beginIndex}" />
                <wcf:param name="sType" value="{WCParm.sType}" />
            </c:when>
        </c:choose>
    </c:if>
</wcf:url>
<div class="page_number">
    <fmt:message key="REVIEW_LIST_PAGE" bundle="{storeText}">
        <fmt:param value="{reviews.pageNum}" />

```



```

        <fmt:param value="\${reviews.noOfTotalPages}" />
    </fmt:message>
</div>
<c:if test="\${reviews.noOfTotalPages > 1}">
    <c:if test="\${reviews.pageNum > 1}">
        <span class="bullet">&#171; </span>
        <fmt:message var="prevTitle" key="REVIEW_LIST_PAGE_PREV_TITLE"
bundle="\${storeText}" />
        <a href="\${fn:escapeXml(ProductReviewListPrevURL)}"
            title="\${fn:escapeXml(prevTitle)}">
            <fmt:message key="REVIEW_LIST_PAGE_PREV" bundle="\${storeText}" />
        </a>
    </c:if>
    <c:if test="\${reviews.pageNum < reviews.noOfTotalPages}">
        <c:if test="\${reviews.noOfTotalPages > 1}">
            &nbsp;
        </c:if>
        <fmt:message var="nextTitle" key="REVIEW_LIST_PAGE_NEXT_TITLE"
bundle="\${storeText}" />
        <a href="\${fn:escapeXml(ProductReviewListNextURL)}"
            title="\${fn:escapeXml(nextTitle)}">
            <fmt:message key="REVIEW_LIST_PAGE_NEXT" bundle="\${storeText}" />
        </a>
        <span class="bullet"> &#187;</span>
    </c:if>
</c:if>
</div>

</c:otherwise>
</c:choose>

```

You cannot modify this view until you complete the steps in 5.5.14, “Modify infrastructure resources” on page 384.

5.5.11 Create a page for review details

In this section, we explain how to create the JSP to shows the review details. You will implement access control, struts configuration, and text bundle configuration at a later stage. Refer to 5.5.14, “Modify infrastructure resources” on page 384 for information about how to configure these files.

To create the `ProductReviewDetails.jsp` file:

1. In the Java EE perspective of IBM WebSphere Commerce V7 Developer Edition, expand and right-click **Stores** \varnothing **WebContent** \varnothing **Madisons** \varnothing **mobile** \varnothing **ShoppingArea** \varnothing **CatalogSection** \varnothing **ReviewSubsection**, and then select **New** \varnothing **File**.
2. The New File wizard opens. Enter `ProductReviewDetails.jsp` in the File name entry field. Then, ensure that the parent folder entry field contains the following path:

Stores/WebContent/Madisons/mobile/ShoppingArea/CatalogSection/Review Subsection

Click **Finish**.
3. The newly created `ProductReviewDetails.jsp` file opens in the source file editor. Enter the skeleton code shown in Example 5-104, and save the file by pressing Ctrl+S.

Example 5-104 Skeleton for ProductReviewDetails.jsp

```
<%--
*****
* This JSP displays the product review details for a given product
*****
--%>

<!-- BEGIN ProductReviewDetails.jsp -->

<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="http://commerce.ibm.com/base" prefix="wcbase" %>
<%@ taglib uri="http://commerce.ibm.com/foundation" prefix="wcf" %>

<%@ include file="../../../include/parameters.jspf" %>
<%@ include file="../../../include/JSTLEnvironmentSetup.jspf" %>

<%-- 1. Retrieve data --%>

<!DOCTYPE html PUBLIC "-//WAPFORUM//DTD XHTML Mobile 1.2//EN"
"http://www.openmobilealliance.org/tech/DTD/xhtml-mobile12.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" lang="${shortLocale}"
      xml:lang="${shortLocale}">
  <head>
```

```

<title>
  <fmt:message key="REVIEW_DETAILS_TITLE" bundle="${storeText}" />:
  <c:out value="${catalogEntry.description.name}" escapeXml="false"/>
</title>
<meta http-equiv="content-type" content="application/xhtml+xml" />
<meta http-equiv="cache-control" content="max-age=300" />
<meta name="viewport" content="width=device-width, initial-scale=1.0,
  user-scalable=no" />
<link rel="stylesheet" href="${cssPath}"/>
</head>

<body>
  <div id="wrapper">

    <%@ include file="../../../include/HeaderDisplay.jspf" %>
    <%@ include file="../../../include/BreadCrumbTrailDisplay.jspf"%>

    <%out.flush();%>

    <div id="review_details" class="content_box">
      <div class="heading_container">
        <h2><fmt:message key="REVIEW_DETAILS_TITLE"
          bundle="${storeText}" /></h2>
        <div class="clear_float"></div>
      </div>

      <%-- 2. Main content --%>

    </div>

    <%out.flush();%>
    <%@ include file="../../../include/FooterDisplay.jspf" %>
  </div>
</body>
</html>

<!-- END ProductReviewDetails.jsp -->

```

The code in Example 5-104 establishes the headers and footers, includes the dependent JSP files, and displays the title text in the main content area.

4. Next, you need to add the code to retrieve the necessary data for the JSP. Locate the following line comment in the code that you inserted in the previous step:

```
<%-- 1. Retrieve data --%>
```

5. Insert the code from Example 5-105 after the line comment, and save the file by pressing Ctrl+S.

Example 5-105 Data retrieval code for ProductReviewDetails.jsp

```
<c:set var="pgGrp" value="${WCPParam.pgGrp}" />
<c:choose>
  <c:when test='${pgGrp == "prodComp"}'>
    <c:set var="prodComparePageGroup" value="true" scope="request"/>
  </c:when>
  <c:when test='${pgGrp == "search"}'>
    <c:set var="searchPageGroup" value="true" scope="request"/>
  </c:when>
  <c:when test='${pgGrp == "wishlist"}'>
    <c:set var="wishlistPageGroup" value="true" scope="request"/>
  </c:when>
  <c:otherwise>
    <c:set var="categoryNavPageGroup" value="true" scope="request"/>
  </c:otherwise>
</c:choose>

<c:set var="productReviewPage" value="true" scope="request"/>
<c:set var="productReviewDetailsPage" value="true" scope="request"/>

<c:set var="productId" value="${WCPParam.productId}" />

<wbase:useBean id="catalogEntry"
classname="com.ibm.commerce.catalog.beans.CatalogEntryDataBean" />

<c:set var="partNumber" value="${catalogEntry.partNumber}" />

<wbase:useBean id="reviews"
classname="com.ibm.itso.commerce.socom.beans.ReviewsDataBean">
  <c:set property="sortOptions" value="rating|desc" target="${reviews}"/>
  <c:set property="partNumber" value="${partNumber}" target="${reviews}"/>
</wbase:useBean>

<wcf:url var="productDisplayLink" value="mProductDisplayView">
  <c:forEach var="parameter" items="${WCPParamValues}">
    <c:forEach var="value" items="${parameter.value}">
      <c:if test='${parameter.key != "reviewId"}'>
        <wcf:param name="${parameter.key}" value="${value}" />
      </c:if>
    </c:forEach>
  </c:forEach>
</wcf:url>

<wcf:url var="reviewListUrl" value="mProductReviewsView">
  <c:forEach var="parameter" items="${WCPParamValues}">
```

```

        <c:forEach var="value" items="${parameter.value}">
            <c:if test="${parameter.key != 'reviewId'}">
                <wcf:param name="${parameter.key}" value="${value}" />
            </c:if>
        </c:forEach>
    </c:forEach>
</wcf:url>

<%-- determine next and previous review for paging --%>
<c:set var="seenThisOne" value="false" />
<c:forEach var="review" items="${reviews.reviewDetailsMap}">
    <c:if test="${seenThisOne && empty nextReview}">
        <c:set var="nextReview" value="${review.value}" />
    </c:if>
    <c:if test="${review.value.reviewId eq WCPParam.reviewId}">
        <c:set var="previousReview" value="${prev}" />
        <c:set var="seenThisOne" value="true" />
    </c:if>
    <c:set var="prev" value="${review.value}" />
</c:forEach>

<wcf:url var="prevReviewDetailsUrl" value="mReviewDetails">
    <c:forEach var="parameter" items="${WCPParamValues}">
        <c:forEach var="value" items="${parameter.value}">
            <c:if test="${parameter.key != 'reviewId'}">
                <wcf:param name="${parameter.key}" value="${value}" />
            </c:if>
        </c:forEach>
    </c:forEach>
    <wcf:param name="reviewId" value="${previousReview.reviewId}" />
</wcf:url>

<wcf:url var="nextReviewDetailsUrl" value="mReviewDetails">
    <c:forEach var="parameter" items="${WCPParamValues}">
        <c:forEach var="value" items="${parameter.value}">
            <c:if test="${parameter.key != 'reviewId'}">
                <wcf:param name="${parameter.key}" value="${value}" />
            </c:if>
        </c:forEach>
    </c:forEach>
    <wcf:param name="reviewId" value="${nextReview.reviewId}" />
</wcf:url>

```

The code in Example 5-105 sets up breadcrumb display variables, retrieves the product ID from the request parameters, instantiates and populates a CatalogEntryDataBean instance to get the part number, and then instantiates and populates a ReviewsDataBean instance to retrieve the review details for the product. Then, the URLs for product display and the review list are

generated for later use. Finally, the code determines the correct URLs to use for paging support.

6. Next, you need to add the overall rating for the product and the button for writing a review and finally the review details themselves. In the source code editor for `ProductReviewDetails.jsp`, locate the following line comment:

```
<!-- 2. Main content -->
```

7. Insert the code from Example 5-106 after this line comment, and press Ctrl+S to save the file.

Example 5-106 Overall rating code for `ProductReviewDetails.jsp`

```
<p>
  <a href="${productDisplayLink}">
    <c:out value="${catalogEntry.description.name}" />
  </a>
</p>
<p><fmt:message key="RATING_OVERALL_TITLE" bundle="${storeText}" />
<%out.flush();%>
<c:import url="${jspStoreDir}mobile/Snippets/ReusableObjects/StarRating.jsp">
  <c:param name="lowerBound" value="${reviews.lowerBound}" />
  <c:param name="upperBound" value="${reviews.upperBound}" />
  <c:param name="rating" value="${reviews.avgOverallRating}" />
</c:import>
<%out.flush();%>
</p>

<%@include file="../../Snippets/Catalog/Reviews/WriteReviewLink.jspf" %>

<c:set var="reviewDetails" value="${reviews.reviewDetailsMap[WCParam.reviewId]}" />

<div class="review_info">
  <%@include file="../../Snippets/Catalog/Reviews/ReviewHeadingDisplay.jspf" %>
</div>

<p><c:out value="${reviewDetails.reviewText}" /></p>

<c:if test="${!(empty nextReview && empty previousReview)}">
  <div class="paging_control">
    <c:if test="${!empty previousReview}">
      <span class="bullet">&#171; </span>
      <a href="${fn:escapeXml(prevReviewDetailsUrl)}"
        title="<fmt:message key='REVIEW_DETAILS_PREVIOUS'
          bundle='${storeText}' />"
        <fmt:message key="REVIEW_DETAILS_PREVIOUS" bundle="${storeText}" />
      </a>
    </c:if>
  </div>
</c:if>
```

```

        </a>
    </c:if>
    <c:if test="${!empty nextReview}">
        <c:if test="${!empty previousReview}">
            &nbsp;
        </c:if>
        <a href="${fn:escapeXml(nextReviewDetailsUrl)}"
            title="<fmt:message key='REVIEW_DETAILS_NEXT'
                bundle='${storeText}' />"
            <fmt:message key="REVIEW_DETAILS_NEXT" bundle="${storeText}" />
        </a><span class="bullet"> &#187;</span>
    </c:if>
</div>
</c:if>

<div class="paging_control">
    <span class="bullet">&#171; </span>
    <a href="${fn:escapeXml(reviewListUrl)}"
        title="<fmt:message key='REVIEW_DETAILS_BACK' bundle='${storeText}' />"
        <fmt:message key="REVIEW_DETAILS_BACK" bundle="${storeText}" />
    </a>
</div>

```

The code in Example 5-106 displays the overall rating along with a link to the Write a Review page, followed by the review details, including the user name of the reviewer, the date the review was submitted, and the actual review text.

5.5.12 Create the JSP for writing the review

You create the PostReview.jsp file to enable user to write a new review for the product. The user must be logged in to write a review. If the user is not logged in, you can display a message and suggest that the user log in to write the review, as shown in Figure 5-52.



Figure 5-52 Write a review page for a guest user

To create the write a review PostReview.jsp:

1. Create a new JSP file named PostReview.jsp in the following directory:
Stores/WebContent/Madisons/mobile/ShoppingArea/CatalogSection/Review
Subsection
2. Begin by creating the skeleton HTML document. Open the JSP editor, and enter the code shown in Example 5-107. Press Ctrl+S to save the file.

Example 5-107 Skeleton code for the PostReview.jsp file

```
<%--
*****
* This JSP takes the review inputs and submits to PostReviewCmd.
*****
--%>

<!-- BEGIN PostReview.jsp -->

<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="http://commerce.ibm.com/base" prefix="wcbase" %>
<%@ taglib uri="http://commerce.ibm.com/foundation" prefix="wcf" %>

<%@ include file="../../../../include/parameters.jspf" %>
<%@ include file="../../../../include/JSTLEnvironmentSetup.jspf" %>
<%@ include file="../../../../include/ErrorMessageSetup.jspf" %>

<%-- 1. Initialize beans and URL links --%>

<!DOCTYPE html PUBLIC "-//WAPFORUM//DTD XHTML Mobile 1.2//EN"
"http://www.openmobilealliance.org/tech/DTD/xhtml-mobile12.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" lang="${shortLocale}"
xml:lang="${shortLocale}">
  <head>
    <title>
      <fmt:message key="WRITE_REVIEW_TITLE" bundle="${storeText}" />:
      <c:out value="${catalogEntry.description.name}" escapeXml="false"/>
    </title>
    <meta http-equiv="content-type" content="application/xhtml+xml" />
    <meta http-equiv="cache-control" content="max-age=300" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0,
user-scalable=no" />
    <link rel="stylesheet" href="${cssPath}"/>
```



```

</head>

<body>
  <div id="wrapper">

    <%@ include file="../../../include/HeaderDisplay.jspf" %>
    <%@ include file="../../../include/BreadCrumbTrailDisplay.jspf"%>

    <%out.flush();%>

    <div id="write_review" class="content_box">
      <div class="heading_container">
        <h2><fmt:message key="REVIEW_WRITE_REVIEW"
          bundle="${storeText}" /></h2>
        <div class="clear_float"></div>
      </div>

      <%-- 2. Main content --%>

    </div>

    <%out.flush();%>
    <%@ include file="../../../include/FooterDisplay.jspf" %>
  </div>
</body>
</html>

<!-- END PostReview.jsp -->

```

- Now, add the code to retrieve product details and to generate the product display page link, as shown in Example 5-108. Add this code *after* the following line from Example 5-107 on page 370:

```

<%-- 1. Initialize beans and URL links --%>

```

Example 5-108 Initialization code for the *PostReview.jsp* file

```

<c:set var="pgGrp" value="${WCPParam.pgGrp}" />
<c:choose>
  <c:when test='${pgGrp == "prodComp"}'>
    <c:set var="prodComparePageGroup" value="true" scope="request"/>
  </c:when>
  <c:when test='${pgGrp == "search"}'>
    <c:set var="searchPageGroup" value="true" scope="request"/>
  </c:when>
  <c:when test='${pgGrp == "wishlist"}'>

```

```

        <c:set var="wishlistPageGroup" value="true" scope="request"/>
    </c:when>
    <c:otherwise>
        <c:set var="categoryNavPageGroup" value="true" scope="request"/>
    </c:otherwise>
</c:choose>

<c:set var="productReviewPage" value="true" scope="request"/>
<c:set var="productReviewPostPage" value="true" scope="request"/>
<c:set var="starImagePath" value="/soccom/ibm/social/images/" />

<wcbase:useBean id="catalogEntry"
    classname="com.ibm.commerce.catalog.beans.CatalogEntryDataBean" />
<wcf:url var="productDisplayLink" value="mProductDisplayView">
    <c:forEach var="parameter" items="${WCParamValues}">
        <c:forEach var="value" items="${parameter.value}">
            <c:if test="${parameter.key != 'reviewId'}">
                <wcf:param name="${parameter.key}" value="${value}" />
            </c:if>
        </c:forEach>
    </c:forEach>
</wcf:url>

<wcf:url var="reviewListUrl" value="mProductReviewsView">
    <c:forEach var="parameter" items="${WCParamValues}">
        <c:forEach var="value" items="${parameter.value}">
            <c:if test="${parameter.key != 'reviewId'}">
                <wcf:param name="${parameter.key}" value="${value}" />
            </c:if>
        </c:forEach>
    </c:forEach>
</wcf:url>

```

4. Add the code for error handling as follows:
 - a. Check for any exceptions thrown from the command.
 - b. If the exceptions are found, check if the exception is because the user is not logged in.
 - c. If the user is not logged in, display the message for the user to register or to log in.
 - d. If the exception is for anything other than this, display the exception message and the write review form.

Example 5-109 shows the code snippet for exception handling.

Example 5-109 Exception Handling for PostReview.jsp

```
<a href="{productDisplayLink}"><c:out
value="{catalogEntry.description.name}" /></a>
<p>
  <span class="field_required_symbol">*</span>
  <fmt:message key="REQUIRED_FIELDS" bundle="{storeText}" />
</p>
<c:choose>
  <c:when test="{!empty storeError.key &&
    storeError.key == '_ERR_USER_NOT_LOGGED_IN'}">
    <wcf:url var="logonURL" value="MobileLogonForm">
      <wcf:param name="catalogId" value="{WCPParam.catalogId}" />
      <wcf:param name="storeId" value="{WCPParam.storeId}" />
      <wcf:param name="langId" value="{WCPParam.langId}" />
    </wcf:url>
    <wcf:url var="registerURL"
      value="MobileUserRegistrationAddForm">
      <wcf:param name="catalogId" value="{WCPParam.catalogId}" />
      <wcf:param name="storeId" value="{WCPParam.storeId}" />
      <wcf:param name="langId" value="{WCPParam.langId}" />
      <wcf:param name="register_button" value="Register" />
    </wcf:url>
    <fmt:message key="WRITE_REVIEW_ERROR" bundle="{storeText}" />
    <fmt:message key="WRITE_REVIEW_REGISTER_OR_LOGIN"
      bundle="{storeText}">
      <fmt:param value="{fn:escapeXml(registerURL)}" />
      <fmt:param value="{fn:escapeXml(logonURL)}" />
    </fmt:message>
  </c:when>
  <c:otherwise>
  <c:choose>
    <c:when test="{!empty errorMessage}">
      <p class="error"><c:out value="{errorMessage}" /></p>
    </c:when>
    <c:otherwise>
      <c:if test="{!empty storeError.key}">
        <p class="error"><c:out value="{storeError.key}" /></p>
      </c:if>
    </c:otherwise>
  </c:choose>
</c:choose>
```

5. Write the code for creating and submitting the Write Review form, as shown in Example 5-110. Insert this code *after* the code from Example 5-109 on page 373. The form submits to action PostReview, which is described in 5.5.13, “Create the command to post the review” on page 375.

Example 5-110 Submission form for the PostReview.jsp file

```
<form method="post" action="PostReview">
<fieldset>
  <p>
    <span class="field_required_symbol">*</span>
    <fmt:message key="SELECT_REVIEW_RATING" bundle="${storeText}" />
  </p>
  <ul class="rating">
    <c:forEach var="item" begin="1" end="5">
      <li>
        <input type="radio" name="rating" value="${item}"
          id="rating_${item}" />
        <fmt:message var="ratingTitle" key="REVIEW_RATING_${item}"
          bundle="${storeText}" />
        <label for="rating_${item}" title="${ratingTitle}">
          <%out.flush();%>
          <c:import
            url="${jspStoreDir}mobile/Snippets/ReusableObjects/StarRating.jsp">
            <c:param name="lowerBound" value="0" />
            <c:param name="upperBound" value="5" />
            <c:param name="rating" value="${item}" />
          </c:import>
          <%out.flush();%>
        </label>
      </li>
    </c:forEach>
  </ul>
  <div class="input_container">
    <div>
      <label for="review_title"><span class="field_required_symbol">*</span>
      <fmt:message key="ENTER_REVIEW_TITLE" bundle="${storeText}" /></label>
    </div>
    <input type="text" id="review_title" name="title" class="coloured_input" />
  </div>
  <div class="textarea_container">
    <div>
      <label for="review_comments"><span class="field_required_symbol">*</span>
      <fmt:message key="ENTER_REVIEW_COMMENTS" bundle="${storeText}" /></label>
    </div>
  </div>
</div>
```

```

        <textarea id="review_comments" name="body" class="coloured_input"
            rows="8"></textarea>
    </div>
    <input type="hidden" name="partNumber" value="${WCPParam.partNumber}"/>
    <c:forEach var="parameter" items="${WCPParamValues}">
        <c:forEach var="value" items="${parameter.value}">
            <input type="hidden" name="${parameter.key}" value="${value}"/>
        </c:forEach>
    </c:forEach>
    <input type="hidden" name="URL" value="mPostReviewsView"/>
    <input type="hidden" name="viewTaskName" value="mPostReviewsView"/>
    <input type="hidden" name="errorViewName" value="mPostReviewsView"/>
    <input type="hidden" name="onPage" value="1"/>
    <input type="hidden" name="sortOptions" value="submissionTime|desc" />
    <input type="submit" id="create_review" name="create_review"
        value="<fmt:message key="REVIEW_SUBMIT" bundle="${storeText}" />"
        class="submit" />
    <input type="button" id="cancel_review" name="cancel_review"
        value="<fmt:message key="REVIEW_CANCEL" bundle="${storeText}" />" />
</fieldset>
</form>

<p><fmt:message key="WRITE_REVIEW_DELAY" bundle="${storeText}" /></p>
</c:otherwise>
</c:choose>

```

This completes the implementation of the `PostReview.jsp` file. You cannot view this page until you complete the steps described in 5.5.14, “Modify infrastructure resources” on page 384.

5.5.13 Create the command to post the review

You need to submit the `PostReview.jsp` file that you created in the previous section to a controller command. In this section, we explain how to create the **`PostReviewCmdImpl`** command to post the reviews to the Social Commerce application. To post a review to the Social Commerce application, the user must have logged in to the mobile storefront and have the `LtpaToken2` cookie in the browser. The `LtpaToken2` cookie is set when a user logs in to the storefront.

Note: The LtpaToken2 cookie is set only when the Social Commerce feature is configured as documented in the WebSphere Commerce Version 7 Information Center:

<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/topic/com.ibm.commerce.install.doc/tasks/tigsoccom.htm>

When running the Social Commerce setup wizard, you can choose whether to use LDAP and LTPA or IBM WebSphere Commerce V7 to generate authentication cookies.

The command for posting reviews needs to submit a JSON object to the Social Commerce application. In the spirit of RESTful APIs, the URI for submitting a review is the same as that for retrieving the list of existing reviews:

`/items/itemId/reviews`

The difference, however, is that the HTTP request method for posting reviews is POST. Along with the request, you need to post a data structure of the syntax shown in Example 5-111.

Example 5-111 Post review data structure

```
{
  "title":{ "type": "string" },
  "body":{ "type": "string" },
  "rating":{ "type": "float" },
  "ItemDescription":{ "type": "string" },
}
```

Example 5-112 shows a sample request for write review query to the Social Commerce application.

Example 5-112 Post review request JSON object

```
{"title":"test review","body":"this is a test
review","rating":"5","itemDescription":"FUL0-01"}
```

Example 5-113 shows a sample response for write review query to the Social Commerce application.

Example 5-113 Post review response JSON object

```
{"status":204,"message":"","data":{"isRatingOnly":false,"reviewId":"123
","reviewText":"this is a test review","userId":null,"title":"test
```

```
review","ItemDescription":"FUL0-01","submissionTime":"2009-08-26T19:01:51.781","rating":5,"itemId":"FUL0-01"]}]}
```

To implement the `PostReviewCmd` interface

1. Create a new package named `com.xxx.commerce.socom.commands`, where `xxx` is the name of your company in the `WebSphereCommerceServerExtensionsLogic` project. We chose the package name `com.ibm.itso.commerce.socom.commands`.
2. Create the `PostReviewCmd` interface in the package. This interface extends the `com.ibm.commerce.command.ControllerCommand` interface. Example 5-114 provides the full code for the interface.

Example 5-114 Source code for `PostReviewCmd`

```
package com.ibm.itso.commerce.socom.commands;

import com.ibm.commerce.command.ControllerCommand;

public interface PostReviewCmd extends ControllerCommand {
    public static final String NAME = PostReviewCmd.class.getName();
    public static final String defaultCommandClassName =
        PostReviewCmdImpl.class.getName();
}
```

3. Create the `PostReviewCmdImpl` implementation class for this interface. The implementation class extends the `com.ibm.commerce.command.ControllerCommandImpl` and implements the `PostReviewCmd` interface that you created in the previous step.
4. Declare the constants and attributes as shown in Example 5-115.

Example 5-115 Constants and attributes for `PostReviewCmdImpl`

```
private static final String CLASS_NAME =
    PostReviewCmdImpl.class.getName();
private static final Logger LOGGER = Logger.getLogger(CLASS_NAME);

private String strPartNumber;
private String strTitle;
private String strBody;
private String strRating;
private Cookie authCookie = null;

public static final ECMessage _ERR_USER_NOT_LOGGED_IN =
    new ECMessage(
        ECMessageSeverity.ERROR, ECMessageType.USER,
```

```

        "_ERR_USER_NOT_LOGGED_IN", "ecUserDefinedMessages");

public static final ECMessage _ERR_CANNOT_WRITE_REVIEW =
    new ECMessage(
        ECMessageSeverity.ERROR, ECMessageType.USER,
        "_ERR_CANNOT_WRITE_REVIEW", "ecUserDefinedMessages");

```

Note: The `ECMessage` instances `_ERR_USER_NOT_LOGGED_IN` and `_ERR_CANNOT_WRITE_REVIEW` messages in Example 5-115 are normally declared in a custom `ECMessage` class that is created for your application. We declared it in `PostReviewCmdImpl` for brevity.

5. Implement the `setRequestProperties` method as shown in Example 5-116. The code retrieves the values of part number, rating, review title, and body parameters as well as the `Ltpa2Token` cookie from the request.

Example 5-116 The `setRequestProperties` method for `PostReviewCmdImpl`

```

public void setRequestProperties(TypedProperty reqParms)
    throws ECApplcationException {
    final String METHOD_NAME = "setRequestProperties";
    LOGGER.entering(CLASS_NAME, METHOD_NAME);

    requestProperties = reqParms;
    strPartNumber = reqParms.getString("partNumber", null);
    strTitle = reqParms.getString("title", null);
    strBody = reqParms.getString("body", null);
    strRating = reqParms.getString("rating", null);

    // we assume that this command is invoked from the HTTP channel
    ViewCommandContext viewCmdCtx =
        (ViewCommandContext)getContext();
    Cookie[] cookies = ((HttpControllerRequestObject)
        (viewCmdCtx.getRequest())).getHttpRequest().getCookies();
    for (int i = 0; i < cookies.length; i++) {
        if (cookies[i].getName().equals("LtpaToken2")) {
            authCookie = cookies[i];
            LOGGER.logp(Level.FINE, CLASS_NAME, METHOD_NAME,
                "Found authentication cookie");
        }
    }
    if (authCookie == null) {
        // User is not logged in
        LOGGER.logp(Level.SEVERE, CLASS_NAME, METHOD_NAME,
            "_ERR_USER_NOT_LOGGED_IN key"

```



```

        +_ERR_USER_NOT_LOGGED_IN.getMessageKey());

        throw new ECApplcationException(
            _ERR_USER_NOT_LOGGED_IN,
            CLASS_NAME,
            METHOD_NAME, "mPostReviewsView");
    }

    LOGGER.exiting(CLASS_NAME, METHOD_NAME);
}

```

6. Implement the `validateParameters` method as shown in Example 5-117. This method validates all the input parameters (part number, rating, title, and body all are required parameters). If any of these parameters is null, an `ECApplcationException` is thrown.

Example 5-117 The `validateParameters` method for `PostReviewCmdImpl`

```

public void validateParameters() throws ECException {
    final String METHOD_NAME = "validateParameters";
    LOGGER.entering(CLASS_NAME, METHOD_NAME);

    if (strPartNumber == null || strPartNumber.equals("")) {
        TypedProperty hshNVPs = new TypedProperty();
        hshNVPs.put("ErrorCode", "20010");
        throw new ECApplcationException(
            ECMessage._ERR_CMD_MISSING_PARAM,
            CLASS_NAME, METHOD_NAME,
            ECMessageHelper.generateMsgParms("partNumber"),
            "mPostReviewsView", hshNVPs);
    }

    if (strRating == null || strRating.equals("")) {
        TypedProperty hshNVPs = new TypedProperty();
        hshNVPs.put("ErrorCode", "20011");
        throw new ECApplcationException(
            ECMessage._ERR_CMD_MISSING_PARAM,
            CLASS_NAME, METHOD_NAME,
            ECMessageHelper.generateMsgParms("rating"),
            "mPostReviewsView", hshNVPs);
    }

    if (strTitle == null || strTitle.equals("")) {
        TypedProperty hshNVPs = new TypedProperty();
        hshNVPs.put("ErrorCode", "20012");
        throw new ECApplcationException(
            ECMessage._ERR_CMD_MISSING_PARAM,

```

```

        CLASS_NAME, METHOD_NAME,
        ECMessageHelper.generateMsgParms("title"),
        "mPostReviewsView", hshNVPs);
    }
    if (strBody == null || strBody.equals("")) {
        TypedProperty hshNVPs = new TypedProperty();
        hshNVPs.put("ErrorCode", "20013");
        throw new ECApplicationException(
            ECMessage._ERR_CMD_MISSING_PARAM,
            CLASS_NAME, METHOD_NAME,
            ECMessageHelper.generateMsgParms("body"),
            "mPostReviewsView", hshNVPs);
    }

    LOGGER.exiting(CLASS_NAME, METHOD_NAME);
}

```

7. Implement the `performExecute` method as shown in Example 5-118. The main logic is delegated to the `executeRequest` and `handleResponse` methods that will be implemented in later steps.

Example 5-118 The `performExecute` code sample

```

public void performExecute() throws ECException {
    final String METHOD_NAME = "performExecute";
    LOGGER.entering(CLASS_NAME, METHOD_NAME);
    try {
        super.performExecute();
        HttpResponse response = executeRequest();
        handleResponse(response);

    } catch (IOException e) {
        LOGGER.logp(Level.SEVERE, CLASS_NAME, METHOD_NAME,
            "IOException: "+e.getMessage());
        throw new ECApplicationException(_ERR_CANNOT_WRITE_REVIEW,
            CLASS_NAME, METHOD_NAME, "mPostReviewsView");
    }
    if (responseProperties == null)
        responseProperties = new TypedProperty();
    responseProperties.remove(ECException.ECMESSAGE);
    requestProperties.remove(ECException.ECMESSAGE);
    responseProperties.put(EConstants.EC_VIEWTASKNAME,
        EConstants.EC_GENERIC_REDIRECTVIEW);
    responseProperties.put(EConstants.EC_URL,
        "mProductReviewsView");
}

```

```

        LOGGER.exiting(CLASS_NAME, METHOD_NAME);
    }

```

8. Implement the `executeRequest` method as shown in Example 5-119.

Example 5-119 Source code for the `executeRequest` method

```

private HttpResponse executeRequest()
    throws UnsupportedOperationException, IOException,
        ClientProtocolException {
    final String METHOD_NAME = "executeRequest";
    LOGGER.entering(CLASS_NAME, METHOD_NAME);
    String urlString = "http://localhost/socom/api/items/"
        + strPartNumber + "/reviews";
    HttpClient client = new DefaultHttpClient();
    client.getParams().setParameter("http.socket.timeout",
        new Integer(5000));
    client.getParams().setParameter("http.protocol.content-charset",
        "utf-8");

    if (LOGGER.isLoggable(Level.FINE)) {
        LOGGER.logp(Level.FINE, CLASS_NAME, METHOD_NAME,
            "URL: {0}", urlString);
    }

    HttpPost method = new HttpPost(urlString);
    String entityString = "{\"title\": \""
        + strTitle + "\", \"body\": \"" + strBody
        + "\", \"rating\": \"" + strRating
        + "\", \"itemDescription\": \"" + strPartNumber + "\"}";

    if (LOGGER.isLoggable(Level.FINE)) {
        LOGGER.logp(Level.FINE, CLASS_NAME, METHOD_NAME,
            "Entity: {0}", entityString);
    }

    HttpEntity requestEntity =
        new ByteArrayEntity(entityString.getBytes("UTF-8"));
    method.setEntity(requestEntity);

    method.getParams().setParameter("http.socket.timeout",
        new Integer(5000));

    method.setHeader("Cookie", "LtpaToken2="+authCookie.getValue());
    method.setHeader("Content-Type",

```

```

        "text/json-comment-filtered; charset=utf-8");
    HttpResponse response = client.execute(method);
    LOGGER.exiting(CLASS_NAME, METHOD_NAME, response);
    return response;
}

```

9. Add the code for `handleResponse` as shown in Example 5-120.

Example 5-120 Source code for `handleResponse`

```

private void handleResponse(HttpResponse response)
    throws IOException, ECAApplicationException {
    final String METHOD_NAME = "handleResponse";
    LOGGER.entering(CLASS_NAME, METHOD_NAME);
    int statusCode = response.getStatusLine().getStatusCode();

    if (LOGGER.isLoggable(Level.FINE)) {
        LOGGER.logp(Level.FINE, CLASS_NAME, METHOD_NAME,
            "HTTP status code: {0}", statusCode);
    }
    HttpEntity entity = response.getEntity();
    String responseString = "";
    String message = "";
    if (entity != null) {
        long len = entity.getContentLength();
        if (len != -1) {
            responseString = EntityUtils.toString(entity);
            try {
                JSONObject json = new JSONObject(responseString);
                message = json.getString("message");
            } catch (JSONException e) {
                LOGGER.logp(Level.SEVERE, CLASS_NAME, METHOD_NAME,
                    "Error generating JSON object", e);
                throw new ECAApplicationException(
                    _ERR_CANNOT_WRITE_REVIEW,
                    CLASS_NAME, METHOD_NAME, "mPostReviewsView");
            }
        }
    }
    if (LOGGER.isLoggable(Level.FINE)) {
        LOGGER.logp(Level.FINE, CLASS_NAME, METHOD_NAME,
            "HTTP response: \"{0}\"", responseString);
    }
    if (statusCode == 404) {

```

```

        if ("ERR_NOT_LOGGED_IN".equals(message)) {
            // User is not logged in
            LOGGER.logp(Level.SEVERE, CLASS_NAME, METHOD_NAME,
                "_ERR_USER_NOT_LOGGED_IN key"
                + _ERR_USER_NOT_LOGGED_IN.getMessageKey());

            throw new EApplicationException(
                _ERR_USER_NOT_LOGGED_IN,
                CLASS_NAME,
                METHOD_NAME, "mPostReviewsView");
        }
        else {
            LOGGER.logp(Level.SEVERE, CLASS_NAME,
                METHOD_NAME, responseString);
            // could not write a review
            throw new EApplicationException(
                _ERR_CANNOT_WRITE_REVIEW,
                CLASS_NAME, METHOD_NAME, "mPostReviewsView");
        }
    }
    LOGGER.exiting(CLASS_NAME, METHOD_NAME);
}

```

10. The `PostReviewCmdImpl` class is almost finished. All you need to do is add the import statements. Add the imports from Example 5-121 to the beginning of the `PostReviewCmdImpl` source file.

Example 5-121 Import statements for `PostReviewCmdImpl`

```

import java.io.IOException;
import java.io.UnsupportedEncodingException;
import java.util.logging.Level;
import java.util.logging.Logger;

import javax.servlet.http.Cookie;

import org.apache.http.HttpEntity;
import org.apache.http.HttpResponse;
import org.apache.http.client.ClientProtocolException;
import org.apache.http.client.HttpClient;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.entity.ByteArrayEntity;
import org.apache.http.impl.client.DefaultHttpClient;
import org.apache.http.util.EntityUtils;
import org.json.JSONException;

```

```
import org.json.JSONObject;

import com.ibm.commerce.command.ControllerCommandImpl;
import com.ibm.commerce.command.ViewCommandContext;
import com.ibm.commerce.datatype.TypedProperty;
import com.ibm.commerce.exception.ECApplicationException;
import com.ibm.commerce.exception.ECException;
import com.ibm.commerce.ras.ECMessage;
import com.ibm.commerce.ras.ECMessageHelper;
import com.ibm.commerce.ras.ECMessageSeverity;
import com.ibm.commerce.ras.ECMessageType;
import com.ibm.commerce.server.ECConstants;
import com.ibm.commerce.webcontroller.HttpControllerRequestObject;
```

11. Open the Stores/JavaSource/Madisons/mobile/storeErrorMessages.properties file, and add error keys from Example 5-122 to the end of the file.

Example 5-122 Custom error messages

```
_ERR_USER_NOT_LOGGED_IN = Please Sign-in to write a review.
_ERR_CANNOT_WRITE_REVIEW = Error while creating review. \
Review not created.
_ERR_CMD_INVALID_PARAM.20010 = Part Number is missing.
_ERR_CMD_INVALID_PARAM.20011 = Select a rating for the review.
_ERR_CMD_INVALID_PARAM.20012 = Enter title for the review.
_ERR_CMD_INVALID_PARAM.20013 = Enter review comments.
```

12. To complete this process, complete the steps in the next section.

5.5.14 Modify infrastructure resources

In this section, you modify the shared resources that need to be changed to support the modified and new pages and commands that you created. You need to complete the following steps:

1. Modify the bread crumb JSP fragment
2. Modify the cascading style sheets
3. Amend the struts configuration file
4. Configure access control
5. Define localized strings for pages

Modify the bread crumb JSP fragment

To display the breadcrumb path properly on the new pages, you must update the JSP fragment that generates the breadcrumb:

1. Open the development environment by selecting **Start** ∅ **Programs** ∅ **IBM** ∅ **WebSphere** ∅ **WebSphere Commerce** ∅ **WebSphere Commerce Developer Enterprise**.
2. Switch to the Java EE perspective if it is not already active:
 - a. Select **Window** ∅ **Open Perspective** ∅ **Other**.
 - b. The Open Perspective window opens. Select **Java EE**, and click **OK**.
3. In the Enterprise Explorer, expand and double-click **Stores** ∅ **WebContent** ∅ **Madisons** ∅ **mobile** ∅ **include** ∅ **BreadCrumbTrailDisplay.jspf**.
4. The file opens in the JSP source code editor. Locate the code shown in Example 5-57.

Example 5-123 Code example

```
<c:when test="\${productPage || storeAvailPage || storeDetailPage}"> <!-- Product display page which is reached by clicking a product on the compare page --%>
```

5. Update the code from Example 5-57 to look like the code in Example 5-58. The inserted code is highlighted in bold.

Example 5-124 Code example

```
<c:when test="\${productPage || storeAvailPage || storeDetailPage || productReviewPage}"> <!-- Product display page which is reached by clicking a product on the compare page --%>
```

6. Locate the code shown in Example 5-125.

Example 5-125 Code example

```
<c:when test="\${storeAvailPage || storeDetailPage}">  
  <c:remove var="bctCatalogEntry" scope="page"/>
```

7. Update the code from Example 5-125 to look like the code in Example 5-126. The inserted code is highlighted in bold.

Example 5-126 Code example

```
<c:when test="\${storeAvailPage || storeDetailPage || productReviewPage}">  
  <c:remove var="bctCatalogEntry" scope="page"/>
```

8. Locate the code shown in Example 5-127.

Example 5-127 Code example

```
<a href="{fn:escapeXml(productDisplayUrl)}"><c:out value="{bctCatalogEntry.description.name}" /></a>
<c:choose>
<c:when test="{storeDetailPage}">
    <wcf:url var="inStoreAvailDetails" value="mInStoreAvailabilityDetailsView">
        <wcf:param name="physicalStoreId" value="{WCPParam.physicalStoreId}"/>
        <wcf:param name="physicalStoreIndex" value="{WCPParam.physicalStoreIndex}"/>
        <wcf:param name="langId" value="{WCPParam.langId}"/>
        <wcf:param name="productId" value="{WCPParam.productId}"/>
        <wcf:param name="itemId" value="{WCPParam.itemId}"/>
        <wcf:param name="catalogId" value="{WCPParam.catalogId}"/>
        <wcf:param name="viewTaskName" value="{WCPParam.viewTaskName}"/>
        <wcf:param name="categoryId" value="{WCPParam.categoryId}"/>
        <wcf:param name="storeId" value="{WCPParam.storeId}"/>
        <wcf:param name="selectedValues" value="{WCPParam.selectedValues}"/>
        <wcf:param name="pgGrp" value="prodComp"/>
    </wcf:url>
    <fmt:message key="DIVIDING_BAR" bundle="{storeText}" />
    <a href="{fn:escapeXml(inStoreAvailDetails)}"><c:out
value="{physicalStores[i].description[0].name}" /></a>
    <fmt:message key="DIVIDING_BAR" bundle="{storeText}" />
    <span class="current_page">
        <fmt:message key="MST_VIEW_MAP" bundle="{storeText}"/>
    </span>
</c:when>
</c:choose>
```

9. Add the code from Example 5-128 *after* the code from Example 5-127.

Example 5-128 Code example

```
<c:when test="{productReviewPage}">
    <c:if test="{productReviewListPage}">
        <fmt:message key="DIVIDING_BAR" bundle="{storeText}" />
        <span class="current_page"><fmt:message key="REVIEW_LIST_TITLE"
            bundle="{storeText}" /></span>
    </c:if>
    <c:if test="{productReviewDetailsPage}">
        <fmt:message key="DIVIDING_BAR" bundle="{storeText}" />
        <a href="{fn:escapeXml(reviewListUrl)}"><fmt:message key="REVIEW_LIST_TITLE"
            bundle="{storeText}" /></a>
        <fmt:message key="DIVIDING_BAR" bundle="{storeText}" />
        <span class="current_page"><fmt:message key="REVIEW_DETAILS_TITLE"
```



```

        bundle="{storeText}" /></span>
    </c:if>
    <c:if test="{productReviewPostPage}">
        <fmt:message key="DIVIDING_BAR" bundle="{storeText}" />
        <a href="{fn:escapeXml(reviewListUri)}"><fmt:message key="REVIEW_LIST_TITLE"
            bundle="{storeText}" /></a>
        <fmt:message key="DIVIDING_BAR" bundle="{storeText}" />
        <span class="current_page"><fmt:message key="WRITE_REVIEW_TITLE"
            bundle="{storeText}" /></span>
    </c:if>
</c:when>

```

10. Locate the code shown in Example 5-129.

Example 5-129 Code example

```

<c:when test="{productPage || wishlistDisplayPage || prodComparePage ||
storeAvailPage || storeDetailPage || shoppingcartDisplayPage}"> <!-- Product display
page which is reached by clicking a product on the search page --%>

```

11. Update the code from Example 5-129 to look like the code in Example 5-130.
The inserted code is highlighted in bold.

Example 5-130 Code example

```

<c:when test="{productPage || wishlistDisplayPage || prodComparePage ||
storeAvailPage || storeDetailPage || shoppingcartDisplayPage || productReviewPage">
<!-- Product display page which is reached by clicking a product on the search page
--%>

```

12. Locate the code shown in Example 5-131.

Example 5-131 Code example

```

<c:when test="{wishlistDisplayPage || prodComparePage || shoppingcartDisplayPage}">
    <c:if test="{wishlistDisplayPage}">
        <fmt:message key="DIVIDING_BAR" bundle="{storeText}" />
        <span class="current_page"><fmt:message key="BCT_WISHLIST"
            bundle="{storeText}" /></span>
    </c:if>
    <c:if test="{shoppingcartDisplayPage}">
        <fmt:message key="DIVIDING_BAR" bundle="{storeText}" />
        <span class="current_page"><fmt:message key="BCT_SHOPPINGCART"
            bundle="{storeText}" /></span>
    </c:if>
    <c:if test="{prodComparePage}">
        <fmt:message key="DIVIDING_BAR" bundle="{storeText}" />
    </c:if>

```

```

        <span class="current_page"><fmt:message key="BCT_PROD_COMPARE_LIST"
            bundle="${storeText}" /></span>
    </c:if>
</c:when>

```

13. Update the code from Example 5-131 to look like the code in Example 5-132. The inserted code is highlighted in bold.

Example 5-132 Code example

```

<c:when test="${wishlistDisplayPage || prodComparePage || shoppingcartDisplayPage ||
productReviewPage}

```

```
<span class="current_page"><fmt:message key="WRITE_REVIEW_TITLE"
    bundle="${storeText}" /></span>
</c:if>
</c:when>
```

14. Locate the code shown in Example 5-133.

Example 5-133 Code example

```
<c:when test="${categoryPage || productPage || wishlistDisplayPage || prodComparePage
|| storeAvailPage || storeDetailPage || shoppingcartDisplayPage}">
```

15. Update the code from Example 5-133 to look like the code in Example 5-134.
The inserted code is highlighted in bold.

Example 5-134 Code example

```
<c:when test="${categoryPage || productPage || wishlistDisplayPage || prodComparePage
|| storeAvailPage || storeDetailPage || shoppingcartDisplayPage ||
productReviewPage}">
```

16. Locate the code shown in Example 5-135.

Example 5-135 Code example

```
<c:if test="${shoppingcartDisplayPage}">
    <fmt:message key="DIVIDING_BAR" bundle="${storeText}" />
    <span class="current_page"><fmt:message key="BCT_SHOPPINGCART"
bundle="${storeText}" /></span>
</c:if>

<c:if test="${storeAvailPage}">
    <fmt:message key="DIVIDING_BAR" bundle="${storeText}" />
    <span class="current_page"><c:out value="${physicalStore.description[0].name}"
/></span>
</c:if>
```

17. Update the code from Example 5-135 to look like the code in Example 5-136.
The inserted code is highlighted in bold.

Example 5-136 Code example

```
<c:if test="${shoppingcartDisplayPage}">
    <fmt:message key="DIVIDING_BAR" bundle="${storeText}" />
    <span class="current_page"><fmt:message key="BCT_SHOPPINGCART"
bundle="${storeText}" /></span>
</c:if>
<c:if test="${productReviewListPage}">
```

```

        <fmt:message key="DIVIDING_BAR" bundle="${storeText}" />
        <span class="current_page"><fmt:message key="REVIEW_LIST_TITLE"
            bundle="${storeText}" /></span>
    </c:if>
    <c:if test="${productReviewDetailsPage}">
        <fmt:message key="DIVIDING_BAR" bundle="${storeText}" />
        <a href="${fn:escapeXml(reviewListUri)}"><fmt:message key="REVIEW_LIST_TITLE"
            bundle="${storeText}" /></a>
        <fmt:message key="DIVIDING_BAR" bundle="${storeText}" />
        <span class="current_page"><fmt:message key="REVIEW_DETAILS_TITLE"
            bundle="${storeText}" /></span>
    </c:if>
    <c:if test="${productReviewPostPage}">
        <fmt:message key="DIVIDING_BAR" bundle="${storeText}" />
        <a href="${fn:escapeXml(reviewListUri)}"><fmt:message key="REVIEW_LIST_TITLE"
            bundle="${storeText}" /></a>
        <fmt:message key="DIVIDING_BAR" bundle="${storeText}" />
        <span class="current_page"><fmt:message key="WRITE_REVIEW_TITLE"
            bundle="${storeText}" /></span>
    </c:if>
    <c:if test="${storeAvailPage}">
        <fmt:message key="DIVIDING_BAR" bundle="${storeText}" />
        <span class="current_page"><c:out value="${physicalStore.description[0].name}"
        /></span>
    </c:if>

```

18. Save and close the file.

Modify the cascading style sheets

To show the new and changed pages correctly, you need to modify the standard cascading style sheet (CSS) file for Madisons Mobile Starter Store:

1. In the Java EE perspective of IBM WebSphere Commerce V7 Developer Edition, expand and double-click **Stores**  **WebContent**  **Madisons**  **mobile**  **css**  **common1_1.css**.

2. The `common1_1.css` file opens in the CSS editor as shown in Figure 5-53.

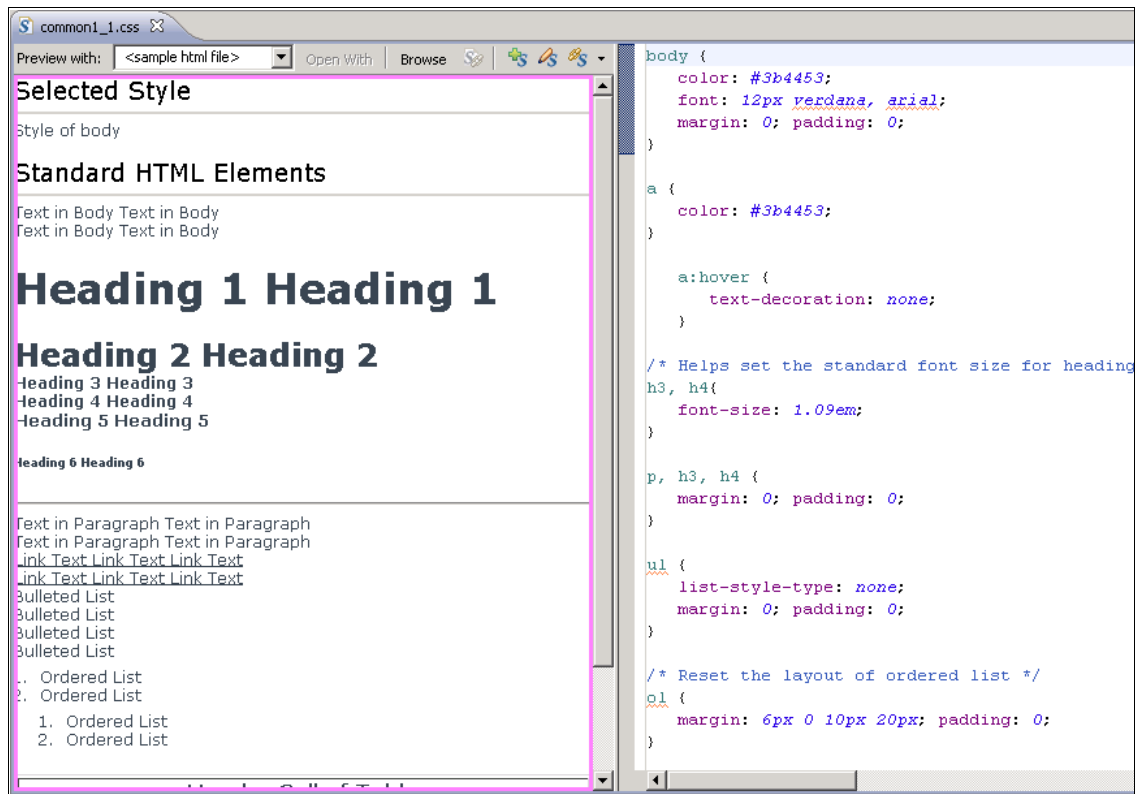


Figure 5-53 The `common1_1.css` file opened in the CSS editor

3. Add the CSS styles shown in Example 5-137 to the bottom of the `common1_1.css` file.
4. Save and close the file by pressing **Ctrl+S** and then **Ctrl+W**.

Example 5-137 New styles for common1_1.css

```

/*****
/*  Social Commerce Styling      */
*****/

```

```
/* Global Product Review Styles */
div#reviews p,
div#review_details p,
div#write_review p {
    margin-top: 9px;
    margin-bottom: 6px;
```

```

}

div#review_details div.paging_control,
div#write_review div.paging_control {
    margin-top: 10px;
}
/*-----*/

/* Reviews */
div#reviews div.sort_by_control {
    border-top: 1px solid #c6d0dc;
    border-bottom: 1px solid #c6d0dc;
    margin: 9px 0; padding: 5px 0;
}

    div.sort_by_control span.current_sort {
        font-weight: bold;
        padding-right: 5px;
    }
/*-----*/

/* Review Details */
div#review_details div.review_info {
    border-top: 1px solid #c6d0dc;
    margin: 9px 0; padding: 5px 0;
}

    div.review_info a.user_image {
        float: left;
    }

    div.review_info ul {
        margin-left: 50px;
    }

        div.review_info ul li {
            padding: 1px 0;
        }
/*-----*/

/* Write review */
div#write_review form {
    border-top: 1px solid #c6d0dc;
    margin: 9px 0;
}

```

```

div#write_review form p.rating {
    margin-left: 8px;
    margin-top: 3px;
}





div#write_review div.textarea_container {
    margin-bottom: 8px;
}

div#write_review div.input_container input,
div#write_review div.textarea_container textarea {
    width: 98%;
}
/*-----*/

```

Amend the struts configuration file

You need to amend the struts configuration file so that it caters to the new pages and command. You need to add global forwards for the new pages and action mappings for the pages and the command. To add the global forwards for the new pages:

1. In the Java EE perspective of IBM WebSphere Commerce V7 Developer Edition, expand and right-click **Stores**  **WebContent**  **WEB-INF**  **struts-config-ext.xml**.
2. In the context menu, select **Open With**  **XML Editor**.
3. The struts-config-ext.xml file opens in an XML editor. Select the **Source** tab if the editor opens in the Design view.
4. Locate the beginning of the global forwards section, as signified by the following line:

```
<global-forwards>
```
5. Insert the forwards shown in Example 5-138 *after* this line.

Example 5-138 Global forwards for struts-config-ext.xml to support mobile ratings and reviews

```

<forward className="com.ibm.commerce.struts.ECActionForward"
    name="mProductReviewsView/10051"
    path="/mobile/ShoppingArea/CatalogSection/ReviewSubsection/ProductReviewList.jsp" />

<forward className="com.ibm.commerce.struts.ECActionForward"
    name="mReviewDetails/10051"
    path="/mobile/ShoppingArea/CatalogSection/ReviewSubsection/ProductReviewDetails.jsp"
/>

```

```
<forward className="com.ibm.commerce.struts.ECActionForward"
    name="mPostReviewsView/10051"
    path="/mobile/ShoppingArea/CatalogSection/ReviewSubsection/PostReview.jsp"/>
```

Note: The forwards in Example 5-138 contain the store ID for the Madisons starter store. In our example, the store ID was 10051. Substitute your value for the Madisons starter store ID when adding the global forwards, as appropriate.

6. Locate the beginning of the action mappings by locating the following line:
`<action-mappings type="com.ibm.commerce.struts.ECActionMapping">`
7. Add the action mappings from Example 5-139 *after* this line.

Example 5-139 Action mappings for struts-config-ext.xml to support mobile ratings and reviews

```
<action
    path="/mProductReviewsView" type="com.ibm.commerce.struts.BaseAction" />

<action
    path="/mReviewDetails" type="com.ibm.commerce.struts.BaseAction" />

<action
    path="/mPostReviewsView" type="com.ibm.commerce.struts.BaseAction" />

<action
    parameter="com.ibm.itso.commerce.socom.commands.PostReviewCmd"
    path="/PostReview" type="com.ibm.commerce.struts.BaseAction" />
```

8. Save and close the file by pressing Ctrl+S and then Ctrl+W.

Configure access control




The IBM WebSphere Commerce access control framework works on an explicit model in which a resource is accessible to a given user only if the access is specifically permitted. In this section, you configure the access control for the new pages and the new command.

To configure the access control policies, you need to complete the following main steps:

1. Create the access control policy file for the new command and views
2. Load the access control policies for the new command and views

Create the access control policy file for the new command and views

To create the access control policy file:

1. Open your favorite text editor. We opened Notepad by selecting **Start**  **Programs**  **Accessories**  **Notepad**.
2. Enter the XML document shown in Example 5-140 into the text editor.

Example 5-140 XML file for acpload to configure access control for mobile ratings and reviews

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no" ?>
<!DOCTYPE Policies SYSTEM "../dtd/accesscontrolpolicies.dtd">

<Policies>

  <Action
    Name="mProductReviewsView"
    CommandName="mProductReviewsView"
  />

  <Action
    Name="mReviewDetails"
    CommandName="mReviewDetails"
  />

  <Action
    Name="mPostReviewsView"
    CommandName="mPostReviewsView"
  />

  <Action
    Name="com.ibm.itso.commerce.socom.commands.PostReviewCmd"
    CommandName="com.ibm.itso.commerce.socom.commands.PostReviewCmd" />

  <ResourceCategory
    Name="com.ibm.itso.commerce.socom.commands.PostReviewCmdResourceCategory"
    ResourceBeanClass="com.ibm.itso.commerce.socom.commands.PostReviewCmd">
    <ResourceAction Name="ExecuteCommand"/>
  </ResourceCategory>

  <ResourceGroup Name="AllSiteUserCmdResourceGroup"
    OwnerID="RootOrganization">
    <ResourceGroupResource
      Name="com.ibm.itso.commerce.socom.commands.PostReviewCmdResourceCategory"/>
  </ResourceGroup>
```

```
<ActionGroup Name="MadisonsAllUsersViews" OwnerID="RootOrganization">
  <ActionGroupAction Name="mProductReviewsView"/>
  <ActionGroupAction Name="mReviewDetails"/>
  <ActionGroupAction Name="mPostReviewsView"/>
  <ActionGroupAction Name="com.ibm.itso.commerce.soccom.commands.PostReviewCmd"/>
</ActionGroup>
</Policies>
```

3. Save the file as `WC_Home\xml\policies\xml\mobile-soccom-acp.xml`. In our example, we saved the file as:

`C:\IBM\WCDE_ENT70\xml\policies\xml\mobile-soccom-acp.xml`

4. Close the text editor.




Load the access control policies for the new command and views

Because you created the access control policy file and placed it in the location to be picked up by IBM WebSphere Commerce, you can load these policies into the development database.

Important: The default database provider for IBM WebSphere Commerce V7 Developer Edition is IBM Cloudscape. IBM Cloudscape allows only one connection to the database at a time. As such, it is not possible to load the access control policies while the server is running.

If you use IBM Cloudscape for the database, you need to stop the server while the access control policies are loaded and then restart the server afterwards.

To load the custom access control policies:

1. Open a command line window by selecting **Start**  **Programs**  **Accessories**  **Command Prompt**.
2. Change to the `WC_Home\bin` directory, where `WC_Home` is the installation directory of IBM WebSphere Commerce V7 Developer Edition. For example, we entered the following command:

```
cd \IBM\WCDE_ENT70\bin
```

3. Run the following command to transform the XML and load the new policies:

```
acpload mobile-soccom-acp.xml
```

Note: If you are not using IBM Cloudscape as the database manager, you need to specify connection parameters on the **acpload** command line. Run **acpload** without any parameters to learn the syntax of the command.

4. The **acpload** command runs for a while. Example 5-141 shows the expected output.

Example 5-141 Expected output from acpload

```
C:\IBM\WCDE_ENT70\bin>acpload mobile-shipping-acp.xml
Running XMLTransform...
Running Id Resolver...
Running MassLoader...
```

Examine `acpload.log` to ensure that everything completed successfully.

```
C:\IBM\WCDE_ENT70\bin>
```

5. Because the output from `acpload` does not notify you of errors that it encounters, you need to inspect the `acpload.log` file for errors, as stated in the console output from `acpload`.

The `acpload.log` file is located in the `WC_Home\logs` directory. In our example, the file was located in the following directory:

```
C:\IBM\WCDE_ENT70\logs
```

The output in `acpload.log` should look similar to the output shown in Example 5-142.




Example 5-142 Output in acpload.log from a successful load

```
Running XMLTransform...
Running Id Resolver...
Running MassLoader...
```

If you are using IBM Cloudscape and have not stopped the server, `acpload.log` contains an error similar to the one shown in Example 5-22. Stop the application that is using the database (most likely the test server) and retry step 3 on page 396.

Define localized strings for pages

The final step that you need to complete to test the modifications is to add the localized texts that are referenced from the JSP code:

1. In the Enterprise Explorer, expand and double-click **Stores** 
Java Resources: JavaSource  **Madisons.mobile** 
storetext.properties.
2. The file opens. Insert the properties from Example 5-143, and press Ctrl+S and then Ctrl+W to save and close the file.

Social Commerce

RATING_OVERALL_TITLE = Overall Rating
RATING_READ_REVIEWS_LINK = Read Reviews
RATING_OVERALL_ERROR = An error occurred while retrieving rating
RATING_OVERALL_NONE = This item has not been rated. \
 Be the first to write a review
WRITE_REVIEW_REGISTER_OR_LOGIN = \
 Register to create a new review or <a href="{1}" \
 title="Sign in">Sign in in if you are already a member.

REVIEW_LIST_TITLE = Reviews
REVIEW_LIST_WRITE_REVIEW = Write a review
REVIEW_LIST_SORT_BY = Sort by:
REVIEW_LIST_SORT_BY_RATING = Highest rating
REVIEW_LIST_SORT_BY_NEWEST = Newest post
REVIEW_LIST_PAGE = Page {0}/{1}
REVIEW_LIST_PAGE_PREV = Previous
REVIEW_LIST_PAGE_PREV_TITLE = Previous page
REVIEW_LIST_PAGE_NEXT = Next
REVIEW_LIST_PAGE_NEXT_TITLE = Next page
REVIEW_OVERVIEW_INFO = By {0} on {1}.
REVIEW_READ_REVIEW = Read Review

REVIEW_DETAILS_TITLE = Review details
REVIEW_DATEFORMAT = M/d/yy h:mma
REVIEW_WRITE_REVIEW = Write a review
REVIEW_DETAILS_NEXT = Next review
REVIEW_DETAILS_PREVIOUS = Previous review
REVIEW_DETAILS_BACK = Back to reviews

PROD_CMPR_RATING = Product Rating
PROD_CMPR_RATING_NONE = (0 reviews)
PROD_CMPR_RATING_ONE = (1 review)
PROD_CMPR_RATING_MULTIPLE = ({0} reviews)

WRITE_REVIEW_TITLE = Write review
WRITE_REVIEW = Write a review
REQUIRED_FIELDS = Denoted required fields
SELECT_REVIEW_RATING = Select rating
REVIEW_RATING_0 = No rating
REVIEW_RATING_1 = Poor (one star)
REVIEW_RATING_2 = Could be better (two stars)

```
REVIEW_RATING_3 = OK (three stars)
REVIEW_RATING_4 = Good quality (four stars)
REVIEW_RATING_5 = Brilliant (five stars)
ENTER_REVIEW_TITLE = Enter a title
ENTER_REVIEW_COMMENTS = Comments
WRITE_REVIEW_DELAY = Content created on this site is subject to a \
    delay in viewing due to administrative reasons.
WRITE_REVIEW_ERROR = Review could not be created.

REVIEW_SUBMIT = Submit
REVIEW_CANCEL = Cancel
```

5.6 Analytics

In this section, we discuss how to use Web Analytics in WebSphere Commerce. We also provide a sample of how to implement page view analysis for the Madisons Mobile Starter Store.

5.6.1 Coremetrics Web Analytics for WebSphere Commerce

The Coremetrics enhanced Web Analytics for WebSphere Commerce provides tools that simplify the process of setting up the site to take advantage of the industry-leading, hosted Web Analytics solution. These tools include a framework that simplifies site configuration, including tagging JSP files so that they provide appropriate analytics information to Coremetrics. You implement the framework as a tag library that is designed specifically to act as an intermediary layer between WebSphere Commerce and Coremetrics.

To take advantage of Coremetrics for IBM WebSphere Commerce, you must install Coremetrics enhanced Web Analytics for WebSphere Commerce and establish a contract with Coremetrics to collect data and to generate and host the resulting reports. The following default reports are available in Coremetrics for WebSphere Commerce:

- ▶ WebSphere Commerce Campaigns Report
- ▶ E-Marketing Spot Report
- ▶ Marketing Experimentation Summary
- ▶ Promotions Summary
- ▶ Business-to-business Contracts
- ▶ Sales Center
 - CSR and Team Summaries
 - CSR Quotas to Orders Conversion Rate

You can view these reports by selecting the **WebSphere Commerce** menu after you have logged on to the Coremetrics site, as showed in Figure 5-54.



Figure 5-54 Access WebSphere Commerce reports from Coremetrics Web site²

With IBM WebSphere Commerce V7, to enable the Coremetrics Web Analytics takes only a few steps:

1. Obtain your Coremetrics client ID and Coremetrics JavaScript libraries from Coremetrics.
2. Update your WebSphere Commerce application in WebSphere Application Server administration console. Select **Replace or add a single file** to add the JavaScript library to the Store Web module path, as shown in Figure 5-55 on page 401.

For more information about updating an application, refer to:

<http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp>

² Screen capture courtesy of Coremetrics, Inc.

Preparing for the application update

Specify the EAR, WAR, JAR, or SAR module to upload and install.

Application to be updated:
WVC

Application update options

☐ Replace the entire application
Upload an enterprise archive (*.ear) to replace the entire installed application.

☐ Replace or add a single module
If the path to the new module matches an existing path to a module in the installed application, the new module replaces the existing module. If the path to the module does not exist in the installed application, the new module is added to the application.

☒ Replace or add a single file
If the path to the new file matches an existing path to a file in the installed application, the new file replaces the existing file. If the path to the file does not exist in the installed application, the new file is added to the application.
Specify the path beginning with the installed application archive file to the file to be replaced or added.

Specify the path to the file.

☒ Local file system
Full path

☐ Remote file system
Full path

☐ Replace, add, or delete multiple files
Use a compressed file format such as .zip or .gzip. The compressed file is unzipped into the installed application directory. If the uploaded files exist in the application with the same paths and file names, the uploaded files replace the existing files. If the uploaded files do not exist, the files are added to the application. You can remove existing files from the installed application by specifying metadata in the compressed file.

Figure 5-55 Update the Application of WebSphere Commerce

3. Modify the biConfig.xml file shown in Example 5-144.

Example 5-144 The biConfig.xml sample file

```
<?xml version="1.0" encoding="UTF-8"?>
<BIConfiguration>
  <biproviders>
    <biprovider name="coremetrics">
      <header>
```

```

        <![CDATA[<script language="JavaScript1.1"
type="text/JavaScript">
<!--
]]>
        </header>
        <footer>
        <![CDATA[
//-->
</script>]]>
        </footer>
        </biprovider>
    </biproviders>

    <stores>
        <store storeId="YourStoreId" biprovider="coremetrics"
enabled="true" debug="true">
            <clientid>YourClientId</clientid>
            <instrumentation>
                <![CDATA[
<script language="JavaScript1.1" type="text/JavaScript"
src="/wcsstore/coremetrics/v40/eluminate.js"></script>
<script language="JavaScript1.1" type="text/JavaScript"
src="/wcsstore/coremetrics/v40/techprops.js"></script>
<script language="JavaScript1.1" type="text/JavaScript"
src="/wcsstore/coremetrics/cmdatatagutils.js"></script>
<script language="JavaScript1.1" type="text/JavaScript">
cmSetProduction();
</script>]]>
                </instrumentation>
            </store>
        </stores>
    </BIConfiguration>

```

Note: You need to change the `YourClientId` parameter to the client ID that you obtained from Coremetrics. You also need to change the `YourStoreId` parameter to the real store ID of the WebSphere Commerce store. Finally, you must uncomment the `cmSetProduction()` line, or you cannot get your results from the Coremetrics Web site.

4. Log on to WebSphere Commerce Accelerator tool, and select the menu **Store** \varnothing **Change Flow** to enable the Coremetrics Web Analytics, as shown in Figure 5-56. Select **Analytics** in the left panel, and then check **Enable Analytics Integration**.

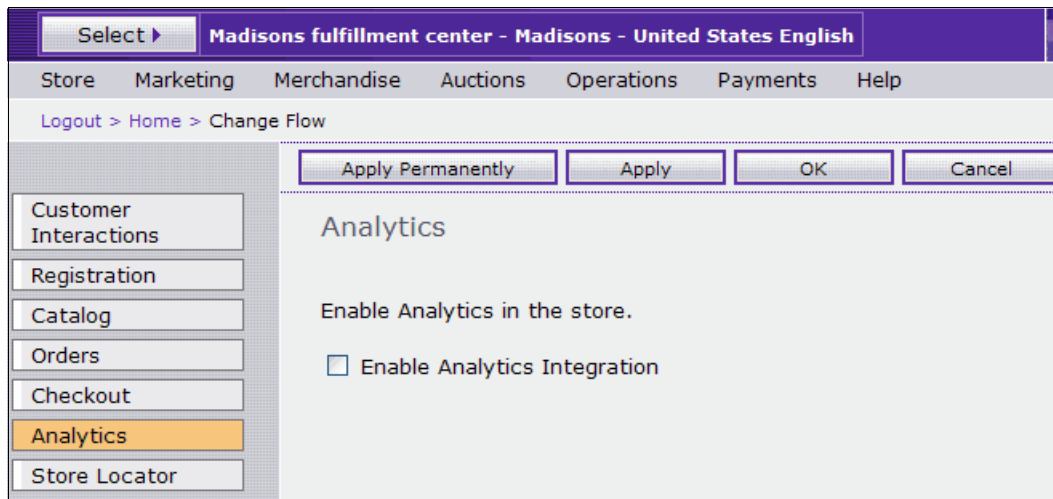


Figure 5-56 Enable Coremetrics Analytics in WebSphere Commerce Accelerator³

5. Run the WebSphere Commerce scenarios. The Coremetrics JSP tags in WebSphere Commerce JSP pages collect the data and send it to Coremetrics for Analytics.
6. Log on to the Coremetrics Web sites to check the analytics results. Coremetrics analyzes the data that it receives every night. You can log on to and view the analytics results one day after they are collected. Figure 5-57 shows an example merchandising report of WebSphere Commerce.

³ Screen capture courtesy of Coremetrics, Inc.

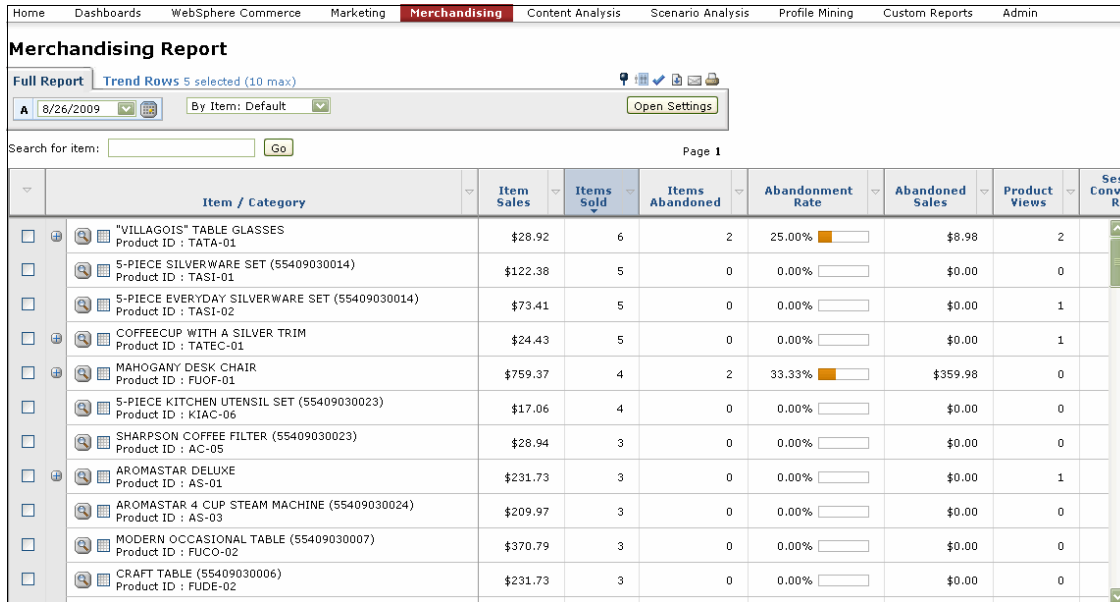


Figure 5-57 Merchandising Report for WebSphere Commerce

5.6.2 Implementing mobile analytics

We give an sample implementation of how to do page view analytics in the Madisons Mobile Starter Store in WebSphere Commerce.

Overview of mobile analytics

By default, Web analytics for WebSphere Commerce Madisons Mobile Starter Store are not enabled in WebSphere Commerce V7. To exploit the business intelligence feature of Coremetrics, you need to take extra steps.

There are different types of pervasive devices that shoppers can use to access the WebSphere Commerce Madisons Mobile Starter Store. For the devices that do not support JavaScript, Coremetrics supplies an image request approach to collect the analytics data. Example 5-145 shows a sample for the image request. The parameters highlighted in bold are Coremetrics parameters.

Example 5-145 A sample image request for devices that do not support JavaScript

```
<img alt="" class="coremetrics"
src=
"/img?url=http%3A%2F%2Fwww.madisonsmobile.ibm.com%2FcreateUID.cfm&titl=1">

```

For more detailed information about the image request approach, refer to:

<http://www.coremetrics.com/>

For devices that do not support the cookies, Coremetrics supplies additional parameters. Refer to Coremetrics for more information if you want to support these types of devices.

For the devices that support JavaScript and cookies, you can use the Coremetrics JSP tags that are supplied by WebSphere Commerce to collect the data.

You can use the following Coremetrics JSP tags in mobile JSP files:

► **<cm:pageview> tag**

You can use the **<cm:pageview>** tag to collect the data for Web pages that are viewed by online shoppers. For information about how to use this tag and its parameters, refer to:

<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.Coremetrics.doc/refs/rmtusepgvw.htm>

► **<cm:order> tag**

You can use the **<cm:order>** tag to collect the order data that is placed by the online shopper in the WebSphere Commerce online store, which must be used in the order confirmation page. For information about how to use this tag and its parameters, refer to:

<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/topic/com.ibm.commerce.Coremetrics.doc/concepts/cmtaglib.htm>

► **<cm:cart> tag**

You can use the **<cm:cart>** tag to collect the content information of the current shopping cart of an online shopper. For information about how to use this tag and its parameters, refer to:

<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.Coremetrics.doc/refs/rmtusecart.htm>

► `<cm:product>` tag

You can use the `<cm:product>` tag to collect the product related information. For information about how to use this tag and its parameters, refer to:

<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.Coremetrics.doc/refs/rmtuseprdt.htm>

► `<cm:registration>` tag

You can use the `<cm:registration>` tag to collect the user registration information. For information about how to use this tag and its parameters, refer to:

<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.Coremetrics.doc/refs/rmtusereg.htm>

► `<cm:campurl>` tag

You can use the `<cm:campurl>` tag to collect the campaign data, which is included in the e-Marketing Spot. For information about how to use this tag and its parameters, refer to:

<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.Coremetrics.doc/refs/rmtusecmp.htm>

► `<cm:contenturl>` tag

You can use the `<cm:contenturl>` tag to collect the content related information in a content spot. For information about how to use this tag and its parameters, refer to:

<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.Coremetrics.doc/refs/rmtusecntspt.htm>

► `<cm:generic>` tag

The `<cm:generic>` tag is available for use as a pass-through tag to communicate additional information about site usage. You can use this tag as a customization tool or to forward information to Coremetrics tags that are not currently supported in the tag library that is provided in Coremetrics for IBM WebSphere Commerce. For information about how to use this tag and its parameters, refer to:

<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.Coremetrics.doc/refs/rmttaggeneric.htm>

Sample mobile analytics implementation

In this sample, we modify the current WebSphere Commerce Madisons Mobile Starter Store JSPs to add Coremetrics JSP tags to do page view analytics with Coremetrics. After the data is sent to Coremetrics and mined, you can log on to the Coremetrics Web site to view the report for the mobile store:

<https://welcome.coremetrics.com/marketforce/Login?auth=2>

To complete this sample:

1. Open the developer's toolkit for IBM WebSphere Commerce V7, as shown in Figure 5-58.

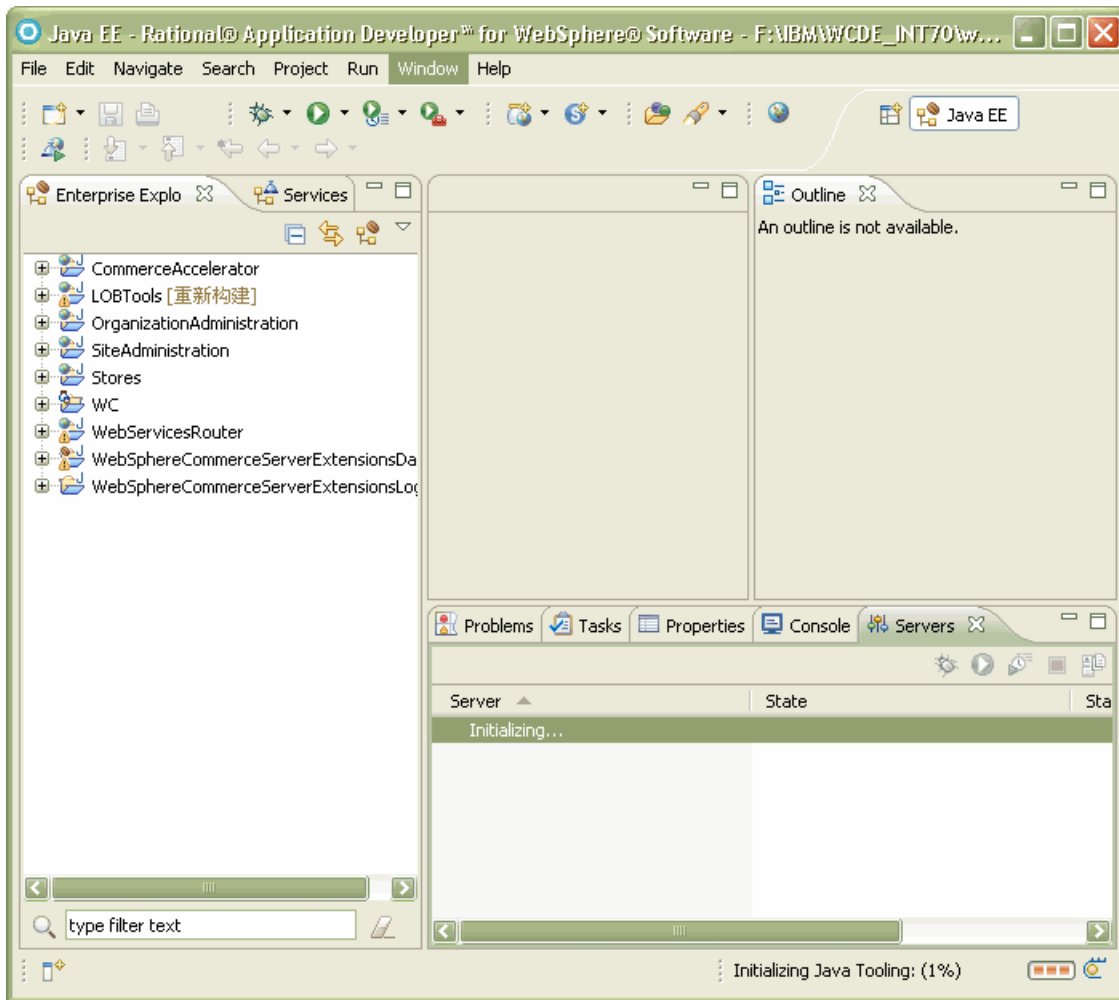


Figure 5-58 WebSphere Commerce V7 toolkit main UI

2. Navigate to the Web project Store, and copy the Coremetrics JavaScript library to the Stores/WebContent directory, as shown in Figure 5-59.

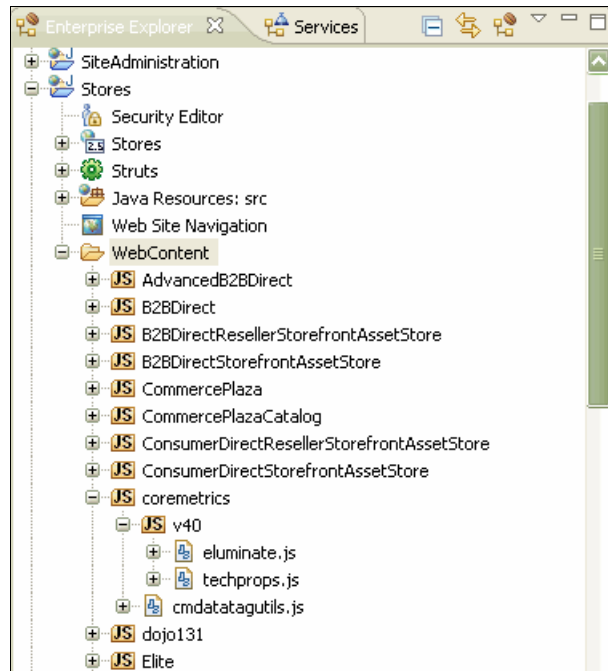


Figure 5-59 Copy the Coremetrics JavaScript library to the Store project

Coremetrics has three JavaScript files:

- eluminate.js
- techprops.js
- cmdatagutils.js

3. Update the biConfig.xml file in the `WCDE_HOME/workspaces/xml/config/bi` directory, as shown in Example 5-144 on page 401.

Note: `WCDE_HOME` is the installation directory for the WebSphere Commerce toolkit (for example `F:\IBM\WCDE_INT70`).

4. Make a backup of the `mobileHome.jsp` file.

5. Locate the following line of code in the `mobileHome.jsp` file:

```
<%@ taglib uri="http://commerce.ibm.com/foundation" prefix="wcf" %>
```

6. Then, insert the following lines of codes *after* this line:

```
<%-- test coremetrics for mobile --%>
<%@ taglib uri="http://commerce.ibm.com/coremetrics" prefix="cm" %>
<%-- test coremetrics for mobile --%>
```

This code imports and declares the Coremetrics tag library of WebSphere Commerce.

7. Locate the HTML `</body>` tag in the `mobileHome.jsp` file, and insert the following lines of code *before* this tag:

```
<%-- test mobile coremetrics begin --%>
<cm:pageview category="SAW912 Mobile" />
<%-- test mobile coremetrics end --%>
```

We inserted the `pageview` tag into this page. We set the `category` parameter to `SAW912 Mobile`, which can be used by Coremetrics to organize the content categories report. Here, we use it as the category to better organize the sample page view results.

All our sample page views are in the same category of `SAW912 Mobile`. We use another import parameter, `pagename`, for the `pageview` tag to specify the name of the page. If no value is set, the default is the page title. We use the default value.

Example 5-146 is the complete `mobileHome.jsp` file after our modification.

Example 5-146 The `mobileHome.jsp` file after modification

```
<%--
=====
Licensed Materials - Property of IBM

WebSphere Commerce

(C) Copyright IBM Corp. 2008, 2009 All Rights Reserved.

US Government Users Restricted Rights - Use, duplication or
disclosure restricted by GSA ADP Schedule Contract with
IBM Corp.
=====
--%>

<%--
*****
* This JSP displays the mobile store home page
*****
```

```

--%>

<!-- BEGIN mobileHome.jsp -->

<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="http://commerce.ibm.com/base" prefix="wcbase" %>
<%@ taglib uri="http://commerce.ibm.com/foundation" prefix="wcf" %>
<%-- test coremetrics for mobile --%>
<%@ taglib uri="http://commerce.ibm.com/coremetrics" prefix="cm" %>
<%-- test coremetrics for mobile --%>
<%@ include file="../include/parameters.jspf" %>
<%@ include file="include/JSTLEnvironmentSetup.jspf" %>

<c:set var="storeId" value="${WCPParam.storeId}" />
<c:set var="catalogId" value="${WCPParam.catalogId}" />

<%-- Required variable for breadcrumb support --%>
<c:set var="mobileIndex" value="true" scope="page" />

<!DOCTYPE html PUBLIC "-//WAPFORUM/DTD XHTML Mobile 1.2//EN"
"http://www.openmobilealliance.org/tech/DTD/xhtml-mobile12.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" lang="${shortLocale}"
xml:lang="${shortLocale}">

    <head>
        <title><fmt:message key="HOME" bundle="${storeText}"/> - <c:out
value="${storeName}"/></title>
        <meta http-equiv="content-type" content="application/xhtml+xml"
/>
        <meta http-equiv="cache-control" content="max-age=300" />
        <meta name="viewport" content="width=device-width,
initial-scale=1.0, user-scalable=no" />
        <link rel="stylesheet" type="text/css" href="${cssPath}" />
    </head>

    <body>

        <div id="wrapper">

            <%@ include file="../include/HeaderDisplay.jspf" %>
            <%@ include file="../include/searchHeader.jspf" %>

            <%--
            <div id="index_banner" class="banner">

```



```

        mobile/images/index_banner.jpg" width="320"
height="75" border="0" alt="Comfort - Relax in luxury" />
        </div>
        <div class="text_container">
            <p>Style your home with these great chairs.</p>
            <p><span class="bullet">&#187; </span><a href="#"
title="More Info">More Info</a></p>
        </div>
        --%>
        <%out.flush();%>
        <c:import
url="\${jspStoreDir}mobile/Snippets/Marketing/ESpot/ContentAreaESpot.jsp
">
            <c:param name="emsName" value="MobileHomePage" />
            <c:param name="catalogId" value="\${WCPParam.catalogId}" />
        </c:import>
        <%out.flush();%>

        <%@ include file="include/BrowseDepartments.jspf" %>
        <%@ include file="include/storeOptions.jspf" %>

        <%out.flush();%>
        <c:import
url="\${jspStoreDir}mobile/Snippets/Marketing/ESpot/FeaturedProductsESpo
t.jsp">
            <c:param name="emsName"
value="MobileHomePageFeaturedProducts" />
            <c:param name="catalogId" value="\${WCPParam.catalogId}" />
            <c:param name="align" value="H" />
        </c:import>
        <%out.flush();%>

        <%@ include file="include/FooterDisplay.jspf" %>
    </div>
    <!-- test mobile coremetrics begin --%>
    <cm:pageview category="SAW912 Mobile" />
    <!-- test mobile coremetrics end --%>

</body>
</html>

<!-- END mobileHome.jsp -->

```

When the shoppers browses this page from a mobile device, data is collected and sent to the Coremetrics server.

8. Repeat step 4 on page 408 through step 7 on page 409 and add the `<cm:pageview>` tag to the following JSP pages.

- ProductDisplay.jsp
- CategoriesDisplay.jsp
- OrderBillingAddressSelection.jsp
- OrderBillingDetails.jsp
- OrderPaymentDetails.jsp
- OrderConfirmationDisplay.jsp
- OrderDetails.jsp
- OrderSummaryDisplay.jsp
- OrderItemDisplay.jsp
- StoreDetails.jsp
- StoreLocator.jsp
- StoreMap.jsp

9. Go to the Madisons Mobile Starter Store home page with your mobile device. Figure 5-60 shows the home page in a Windows mobile simulator.

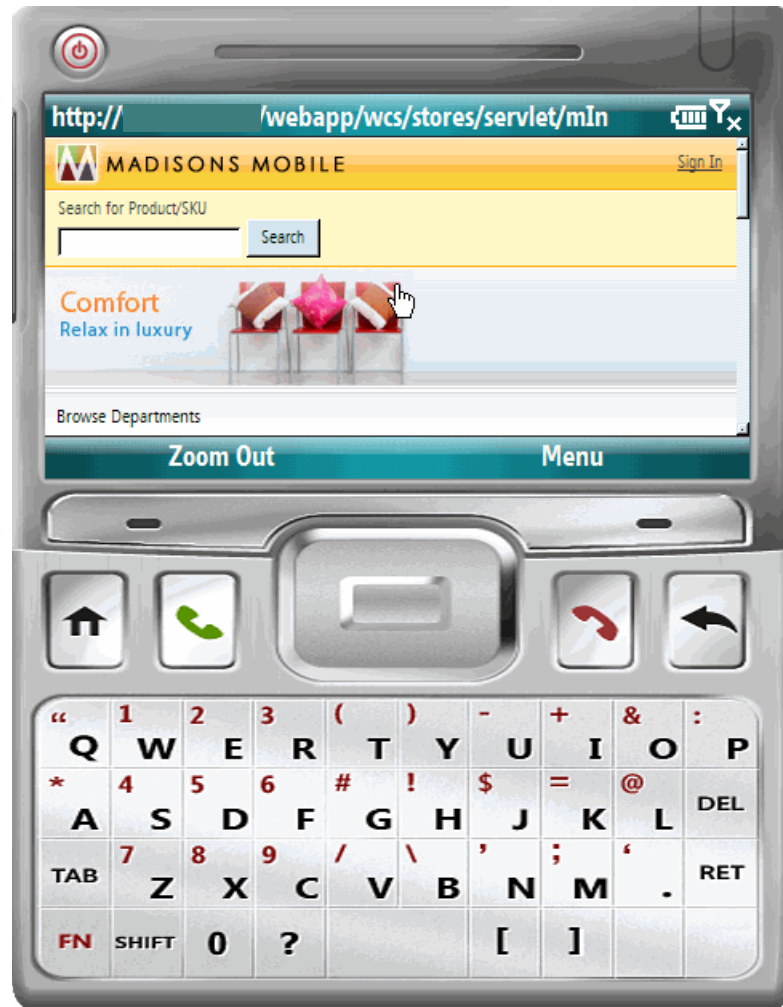


Figure 5-60 Madisons Mobile Starter Store home page

Scroll to the bottom of the home page to see the following line as shown in Figure 5-61:

```
cmCreatePageviewTag(document.title, "SAW912 Mobile", null, null, "10101");
```

This line of code was added by the <cm:pageview> tag, which collects data and sends the data to the Coremetrics server.

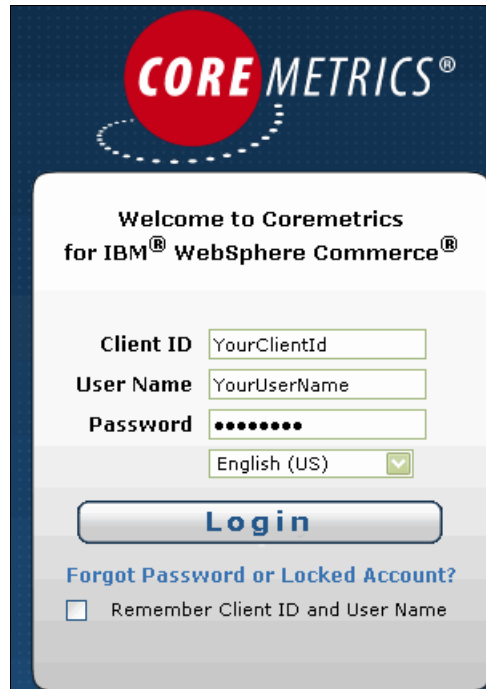


Figure 5-61 Mobile Store Home Page bottom

10. Browse more pages of the mobile store or place an order. You can refer to 4.2.2, "BOPIS in Madisons Starter Store" on page 135 for more detailed steps

for placing an order. There is only slight difference between the Madisons Starter Store shopping flow and Madisons Mobile Starter Store shopping flow.

11. Log on to the Coremetrics Web site with your client ID, user name, and password, as shown in Figure 5-62.



The image shows a login form for Coremetrics. At the top is the Coremetrics logo, which consists of a red circle with the word "CORE" in white and "METRICS" in white to its right. Below the logo is a white box with a dark blue border. Inside the box, the text "Welcome to Coremetrics for IBM® WebSphere Commerce®" is displayed. Below this text are four input fields: "Client ID" with the placeholder text "YourClientId", "User Name" with the placeholder text "YourUserName", "Password" with a series of dots, and a language dropdown menu currently set to "English (US)". Below the input fields is a blue "Login" button. Below the button is a link that says "Forgot Password or Locked Account?". At the bottom of the form is a checkbox labeled "Remember Client ID and User Name".

Figure 5-62 Coremetrics login⁴

Note: You need to perform this step one day after you complete the shopping flows in the Madisons Mobile Starter Store. It takes one day for Coremetrics to mine the data and generate the reports. For more information about working with Coremetrics analytics, refer to:

<http://www.coremetrics.com/>

⁴ Screen capture courtesy of Coremetrics, Inc.

12. In the Coremetrics home page, default reports are displayed. We do not use these reports, so you can click **Content Analysis** menu, as shown in Figure 5-63.

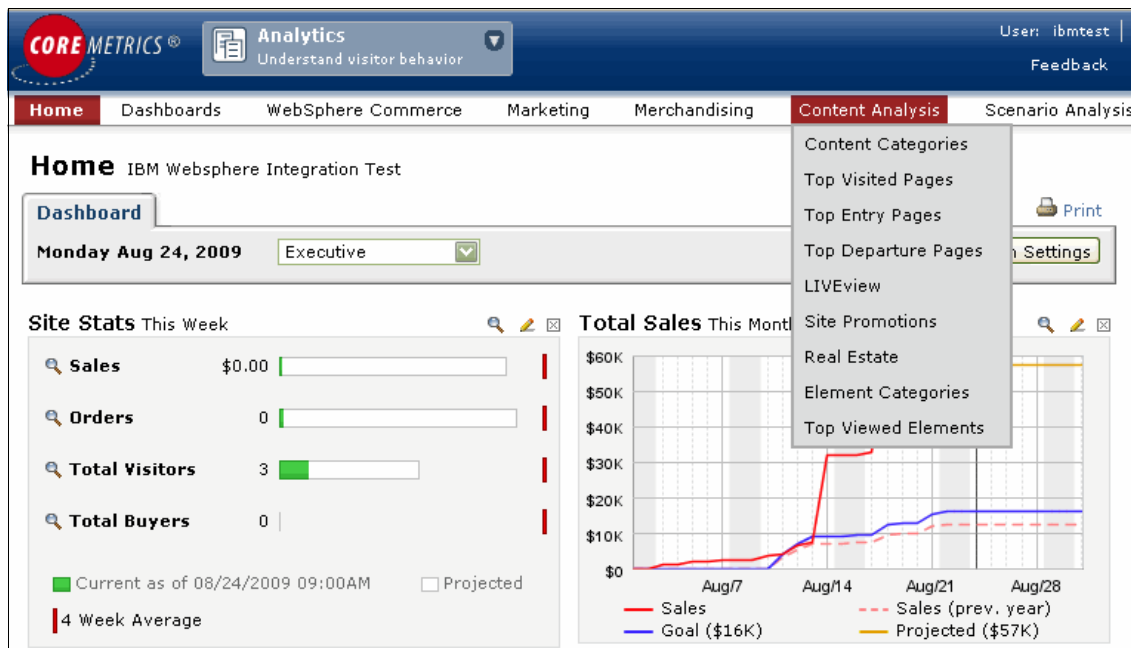


Figure 5-63 Select Content Analysis menu in Coremetrics home⁵

⁵ Screen capture courtesy of Coremetrics, Inc.

13. Select **Content Categories** menu item, the Coremetrics report of content categories is displayed, as shown in Figure 5-64.

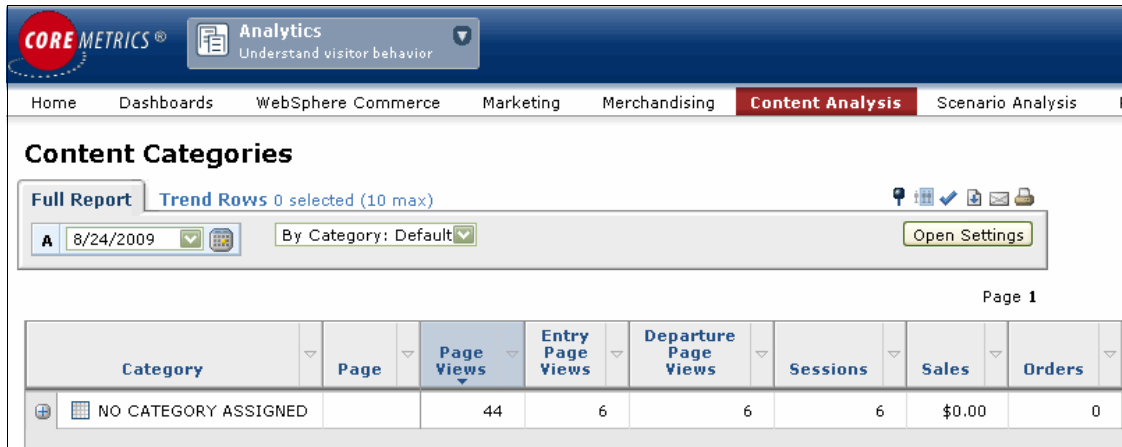


Figure 5-64 Content categories report, collapsed⁶

14. Expand the category of NO CATEGORY ASSIGNED, as shown in Figure 5-65. The tagged category of SAW912 Mobile is displayed.

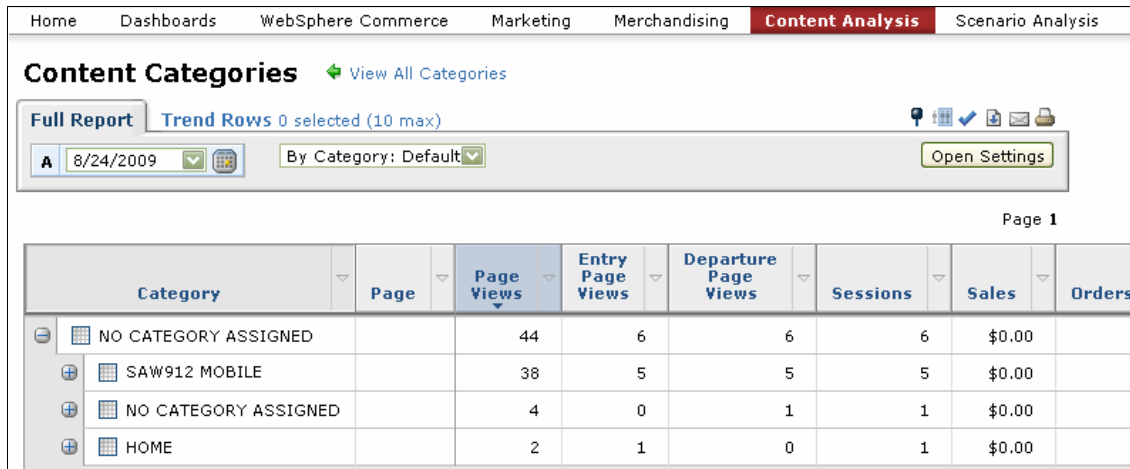


Figure 5-65 Content categories report with SAW912 Mobile collapsed⁷

⁶ Screen capture courtesy of Coremetrics, Inc.

⁷ Screen capture courtesy of Coremetrics, Inc.

15. Expand the **SAW912 Mobile** category. The detailed report contains all the information about the pages the shopper viewed, as shown in Figure 5-66.

Category	Page	Page Views	Entry Page Views	Departure Page Views	Sessions
NO CATEGORY ASSIGNED		44	6	6	
SAW912 MOBILE		38	5	5	
SAW912 MOBILE	HOME - MADISONS Page ID : HOME - MADISONS	14	5	4	
SAW912 MOBILE	ORDER SUMMARY - MADISONS Page ID : ORDER SUMMARY - MADISONS	4	0	0	
SAW912 MOBILE	SHOPPING CART - MADISONS Page ID : SHOPPING CART - MADISONS	4	0	0	
SAW912 MOBILE	PRODUCT: RED LEATHER ROLL ARM CHAISE Page ID : PRODUCT: RED LEATHER ROLL ARM CHAISE	4	0	0	
SAW912 MOBILE	ORDER CONFIRMATION - MADISONS Page ID : ORDER CONFIRMATION - MADISONS	3	0	1	
SAW912 MOBILE	PAYMENT - MADISONS Page ID : PAYMENT - MADISONS	3	0	0	
SAW912 MOBILE	BILLING ADDRESS - MADISONS Page ID : BILLING ADDRESS - MADISONS	2	0	0	
	YOUR BILLING ADDRESSES - MADISONS				


Figure 5-66 Detailed report of SAW912 Mobile category⁸

The report contains the columns of the category name, the page name, page view counts, entry page view counts, departure page view counts, sessions. Detailed information about these report columns are listed below:

- ▶ Category name displays the name of the category.
- ▶ Page name displays the name of the page.
- ▶ Page view displays the counts of one page.
- ▶ Entry page view displays the counts of the page that is taken as the entry point of a user session.
- ▶ Departure page view displays the counts of the page which is taken as the exit point of a user session.
- ▶ Sessions displays the session counts of the page.

For example, the HOME MADISONS page has been viewed 14 times. Among these 14 page views, it was viewed five times as the entry point of a session and four times as the departure of a session. In addition, the page was accessed by five different sessions. One user session placed an order and accessed the order confirmation page, which is displayed by the ORDER CONFIRMATION page.

⁸ Screen capture courtesy of Coremetrics, Inc.

16. Coremetrics also supplies top ranking information. Select **Content Analysis**  **Top Visited Pages** to display the top visited pages, as shown in Figure 5-67.








































Page	Unique Visitors	Page Views	Average Time on Page
   HOME - MADISONS	2	14	0:00:20
   PRODUCT: RED LEATHER ROLL ARM CHAISE	1	4	0:00:50
   SHOPPING CART - MADISONS	1	4	0:00:54
   ORDER SUMMARY - MADISONS	1	4	0:00:23
   ORDER CONFIRMATION - MADISONS	1	3	0:00:15
   PAYMENT - MADISONS	1	3	0:00:30
   HOME - MADISONS MOBILE	1	2	0:00:51
   MADISONS MOBILE - TABLE LAMPS - CATEGORY	1	2	0:00:12
   BILLING ADDRESS - MADISONS	1	2	0:00:38
   YOUR BILLING ADDRESSES - MADISONS	1	2	0:00:08
   PRODUCT: WHITE FABRIC LOVESEAT	1	1	0:00:05
   MADISONS MOBILE - FURNITURE - CATEGORY	1	1	0:00:37
   MADISONS MOBILE - PR...CHA LIME TABLE LAMP	1	1	0:01:08

Figure 5-67 Top visited pages report⁹

The top visited pages report shows the top 1,000 pages that were viewed at least once in the time period. The report contains columns of page, unique visitors, page views, and average time on page.


For more Coremetrics report types, refer to:

<http://www.coremetrics.com/>

You can add other tags to the Madisons Mobile Starter Store to perform more analytics, such as the merchandising and business-to-business contract analytics. For more information, refer to IBM WebSphere Commerce Version 7 Information Center:

<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp>

⁹ Screen capture courtesy of Coremetrics, Inc.



Example client cross-channel solution

This chapter shows how to apply IBM WebSphere Commerce features to a cross-channel solution. The Easy Hogar y Construcción company that we describe in this chapter scales from a service-oriented architecture (SOA) store to a cross-channel business model. This chapter discusses targeted marketing and promotion for Web and mobile channel users as well as how to use this information from an analytical perspective.

6.1 Sample client solution

Note: The Easy Hogar y Construcción company is a real company, not a fictional company.

Easy Hogar y Construcción is a home improvement retail store in South America. This company is owned by one of the biggest retailers in Latin America and has a variety of retail formats such as grocery stores, department stores, specialty stores, travel agencies, malls, and recreation centers among others. Easy Hogar y Construcción products include appliances, building materials, tools and hardware, paint, storage, outdoors, kitchen, flooring, decor, doors and windows, electrical, lighting and fans, and so forth.

A number of years ago, Easy Hogar y Construcción decided to open a new store in Colombia. The company's board realized the benefits of trying a new retail model with this store. The new store provided an opportunity for Easy Hogar y Construcción to move from an SOA store to a cross-channel solution. This new solution created a shared business vision and a sense of ownership between the company and the business.

Figure 6-1 shows a one of the Easy Hogar y Construcción company brick-and-mortar stores.



Figure 6-1 Easy Hogar y Construcción

6.1.1 Challenges for today's global retailer

As a global retailer, Easy Hogar y Construcción must take advantage of common processes in order to maintain a regional presence while also adapting to local demands and expectations.

Over the last 6 years, the retail industry has become more operationally focused and resources have gone into supply chain efficiency, cost reduction, and operational effectiveness. Focusing resources on these tasks was driven primarily by pressure from competitors; however, Easy Hogar y Construcción understands that today's shoppers want convenience and product availability. The company must be ready for this market's opportunity.

Today's retailers also face increasing pressures due to the weak economic environment, which increases costs and competition from new sources.

Note: Being a successful company today is not just about saving time and money. Companies must realize that the shopping experience begins and ends at the shopper's front door, not the front door of a brick-and-mortar store.

The major concern for Easy Hogar y Construcción was that touchpoints for shoppers were separate silos of execution, both organizationally and technologically; however, the shopper sees retail brand identity as a single entity.

6.1.2 Business requirements

The new cross-channel solution for Easy Hogar y Construcción includes the following main business requirements:

- ▶ Design a retail model that lets the company grow and operate efficiently.
- ▶ Identify high service hours that occur because of shopper complaints, and invest in solutions to address the issues to achieve the following goals:
 - High customer satisfaction
 - Improved profit margin
 - Increased revenue potential
 - Additional customer reach
 - Low cost transactions
 - Personalized customer service
 - Optimal buyer impulse use (precision marketing)
 - Access to real-time information
 - Increased transaction size (average ticketing)
 - Enhanced loyalty
 - New channel support with common services

- An intuitive, inviting, and fast shopping experience
- Real time to market
- Consumer activity tracking that provides a single view of the shopper

6.1.3 IT requirements

In keeping with the business goals outlined in 6.1.2, “Business requirements” on page 424, the IT department of Easy Hogar y Construcción suggests the following technical requirements for this solution:

- ▶ Decouple business systems to allow reuse of business logic in other environments.
- ▶ Reduce IT development costs by adhering to technical standards for the retail industry.
- ▶ Lower maintenance costs by integrating disparate applications, databases, and operating systems.
- ▶ Take advantage of the network and Internet technology to support shopper demand and to meet shopper expectations.

6.1.4 Analyzing the requirements and managing expectations

To determine how to proceed to meet the requirements of a successful cross-channel solution, Easy Hogar y Construcción defined business scenarios with relevant stakeholders. Easy Hogar y Construcción began by asking questions such as “How much of our existing technology assets are viable in the new cross-channel efforts?” Using risk analysis, the company determined which requirements had a high negative impact to the business and needed to be delayed, and which requirements failed a cost benefit analysis (CBA) and needed to be rejected. After the risk analysis, the company then communicated the requirements throughout the company to ensure that expectations were met successfully.

Easy Hogar y Construcción began with an enterprise architecture approach using retail industry standards that let the company identify the gaps for building a roadmap based on actual IT projects. The company defined business and technology standards capabilities that were required to realize its goals. The first steps included:

- ▶ Assessing present operations.
- ▶ Defining future capabilities to enable a new efficient retail model that includes a cross channel strategy using best practices.

- ▶ Assessing the ability to execute a strategy that includes costs and risk levels.
- ▶ Defining and prioritizing gaps.

Then, Easy Hogar y Construcción made a high-level strategic roadmap and business case where they defined key projects, costs, and time lines that are required to achieve the following objectives:

- ▶ Develop a multi-stage roadmap for maximum value realization
- ▶ Identify opportunities for quick wins
- ▶ Pinpoint milestones to gauge progress
- ▶ Establish new business metrics and expected return on investment

The Easy Hogar y Construcción multi-stage roadmap provides the following main tasks:

- ▶ Aligning the business and IT processes using SOA
- ▶ Building a cross-channel strategy

We describe these tasks in the sections that follow.

6.1.5 Phase 1: Aligning the business and IT processes using SOA

Easy Hogar y Construcción uses IBM Retail Integration Framework and the Association for Retail Technology Standards (ARTS) to build the basic framework to use SOA as the main vehicle for integration within the company's IT infrastructure. Using a bottom-up approach and service-oriented modelling and architecture (SOMA) methodology, Easy Hogar y Construcción identified services for applications and main processes to build a main catalog that enables business applications to integrate according to the company's business strategy.

For better interoperability, the company decided that all applications must use the ARTS XML standard interface. Also, Easy Hogar y Construcción decided to use the ARTS SOA blueprint that defines how Web services standards are joined with the ARTS message and data standards model to build a retail SOA. This model defines major retail enterprise business levels such as

- ▶ Home office
- ▶ Distribution center
- ▶ Retail store business level

Figure 6-2 describes the main components for an SOA blueprint in the retail industry. This model offers a good reference to develop an enterprise architecture for retail companies that uses a common vocabulary between the business and IT process. It provides business scenarios, Web services samples, data models, and XML schemas that the company can use following the IBM

Retail Integration Framework to build a solid foundation for the business and IT environment.

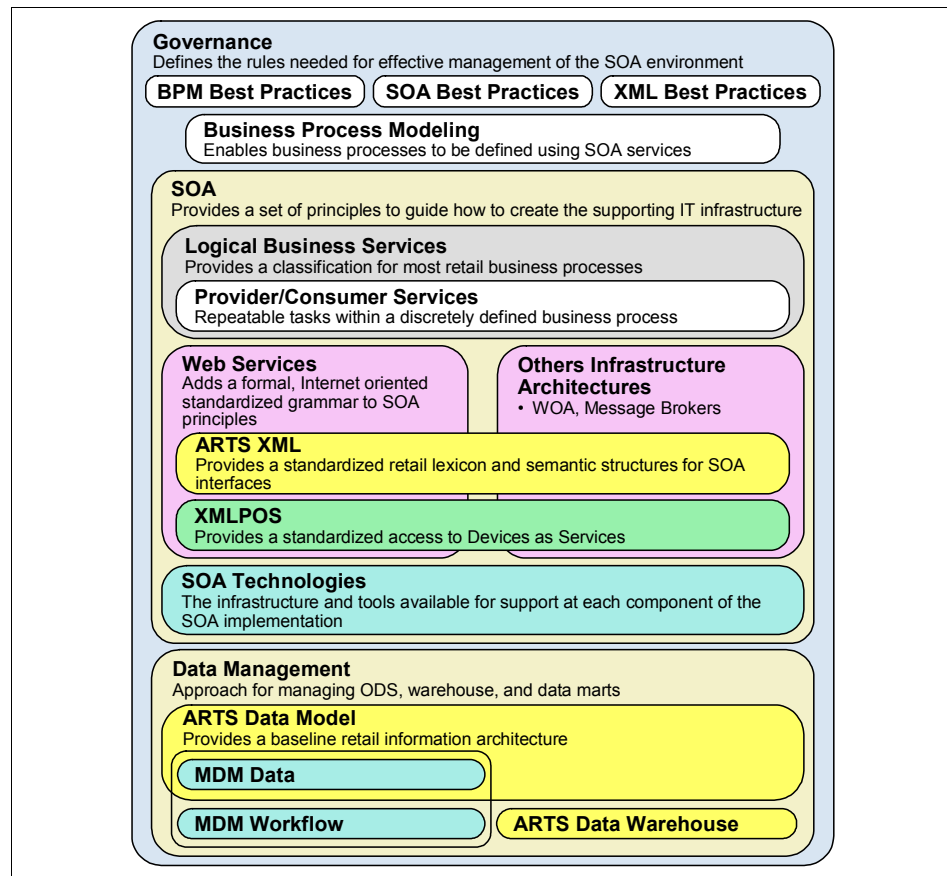


Figure 6-2 ARTS SOA blueprint and best practices overview

For Easy Hogar y Construcción, using ARTS with the IBM Retail Integration Framework provides the following benefits:

- ▶ A common language for all platforms.
- ▶ All applications are decoupled, and Easy Hogar y Construcción can model business processes while interconnecting actual applications.

- ▶ All devices are ready to use a service from a business process or application, so shoppers can obtain accurate information from Easy Hogar y Construcción.
- ▶ Point-of-Sale (POS) systems, PDAs, and kiosks are 100% SOA enabled, so they use services to get shopper information, promotions, tax calculations, orders, and so forth from one single service repository where governance policies are applied.

Figure 6-3 shows how this SOA approach can be applied.

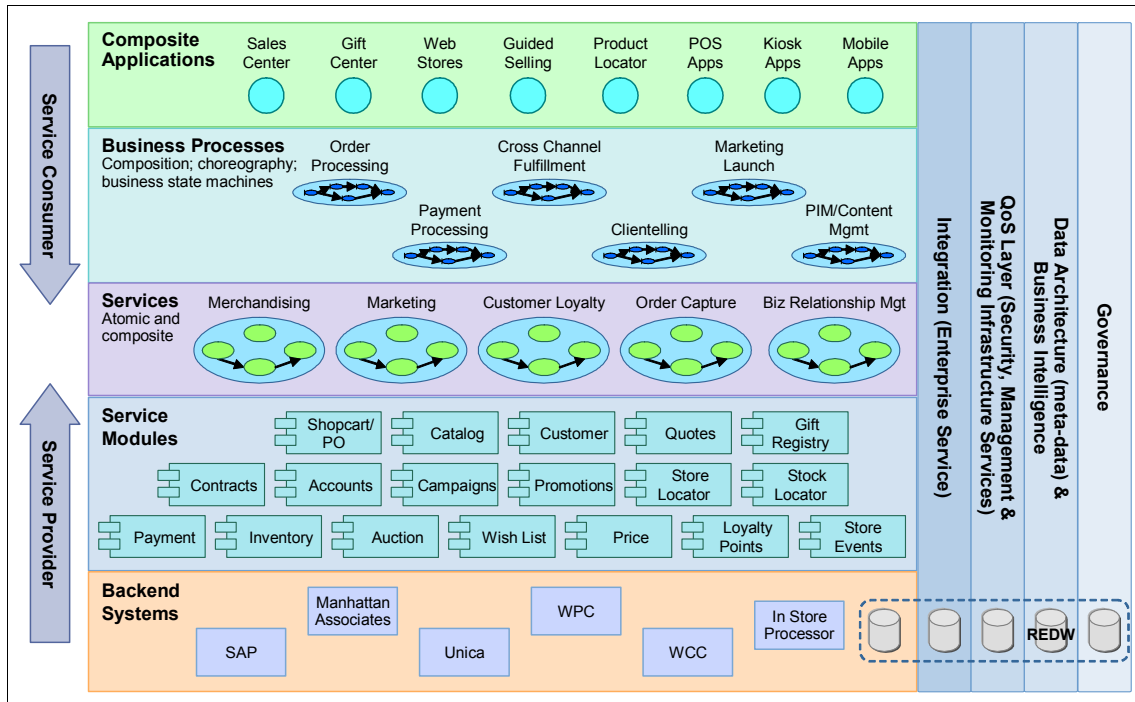


Figure 6-3 SOA logical model

SOA store model logical components

The architecture for an SOA store model includes the following components:

- In-store component

This component is usually called *in-store processor*. It is a server in the store. This server has several devices connected to it, including POS controllers and terminals, kiosks, self-checkout systems, and other devices.

- SOA-enabled POS

POS applications use store integrator capabilities to extend or bypass the logic of existing POS applications using a central SOA server to communicate with the Web and a variety of other applications and systems to support mobile shoppers, Web-based sales transactions, and a variety of devices.

- Enterprise component

This component is a central management point for the in-store processors. Software installations and monitoring data is managed from the Enterprise server.

Figure 6-4 shows a high-level retail environment sample configuration.

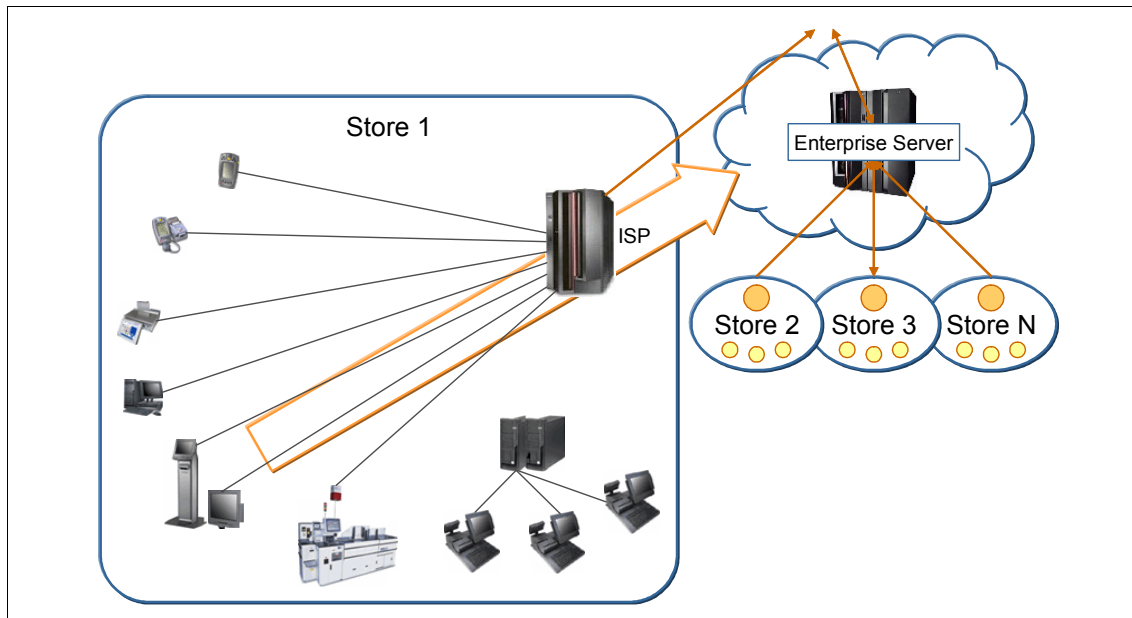


Figure 6-4 Sample retail environment

This environment consists of the following components:

- ▶ Enterprise service bus
A component to manage integration and connectivity logic to de couple applications.
- ▶ Service registry
A component to manage SOA governance.

SOA store model software components

The SOA store model solution include the following software components:

- ▶ IBM WebSphere Remote Server
This solution is a collection of IBM middleware products that is intended to be a software stack for running Web applications within a store called *WebSphere Remote Server Entry Edition* and at main datacenter called *WebSphere Remote Server Enterprise Edition*. It includes the following components:
 - IBM WebSphere Application Server
 - IBM WebSphere MQ
 - IBM DB2
 - Tivoli Configuration Manager
 - Tivoli Enterprise Console® (TEC)
 - Tivoli Monitoring
 - Tivoli Monitoring for Databases
 - IBM Tivoli Composite Application Manager for WebSphere
 - Tivoli OMEGAMON® XE for Messaging
- ▶ IBM WebSphere Systems Management Accelerators
These accelerators are a collection of tools that provide multiple functions in setting up a store environment. It provides the following features:
 - Provides easy installation of WebSphere Remote Server across numerous machines.
 - Extends the monitoring capability of IBM Remote Management Agent to collect information from events at store.
 - Provides integration between WebSphere Remote Server components running at the ISP and Tivoli products running at the enterprise central site.
- ▶ IBM ACE POS
ACE is the next generation IBM POS that takes advantage of traditional IBM POS solutions with new integration components that are coded in C++ and Java.

► Store Integrator Application Extension Facility (AEF)

This solution lets the POS terminal sales program run under the AEF client session server software in a POS terminal, on an in-store processor, or in any device as part of a Web-based application. It includes the following main components:

- POS Data Provider, which monitors events and only provides information
- POS Automation Provider, which allows an external application to use POS functions and is driven by an API
- POS Service Provider Proxy, which allows an external application to add or replace POS functions and is used to replace or alter POS behavior
- POS Device Access, which provides control of the POS I/O

Figure 6-5 shows the ACE development environment, where Easy Hogar y Construcción developers can trace transaction flow and analyze data storage. Using AEF, it is possible to add new dialog boxes or to invoke new services to enhance POS functionality.

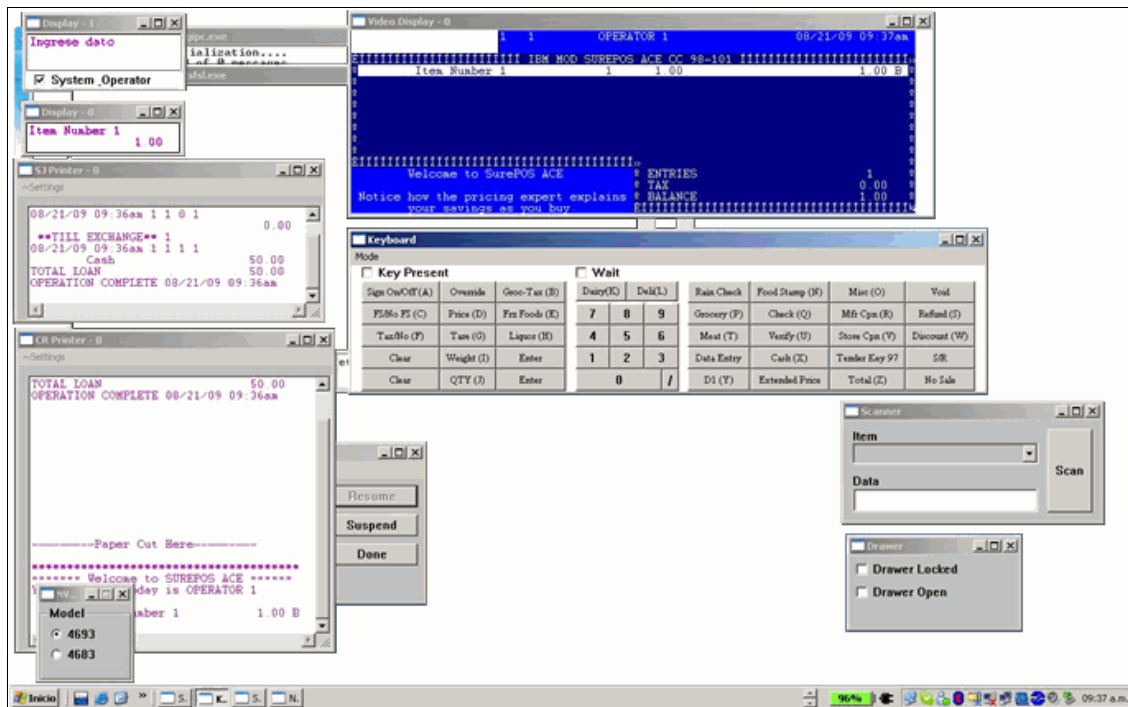


Figure 6-5 ACE development environment

► Store Integrator GUI

Using this feature with AEF provides touch-screen support for operator and customer graphical interfaces. The data transferred between the existing application and Store Integrator GUI is in XML format, rather than being unique to the particular POS application being used. Store Integrator GUI runs on top of AEF, which in turn runs on top of the existing terminal sales application. The flexibility of this architecture enables AEF and Store Integrator GUI to run without requiring modifications to the terminal sales application.

Figure 6-6 shows store integrator flows to manage a sales transaction.

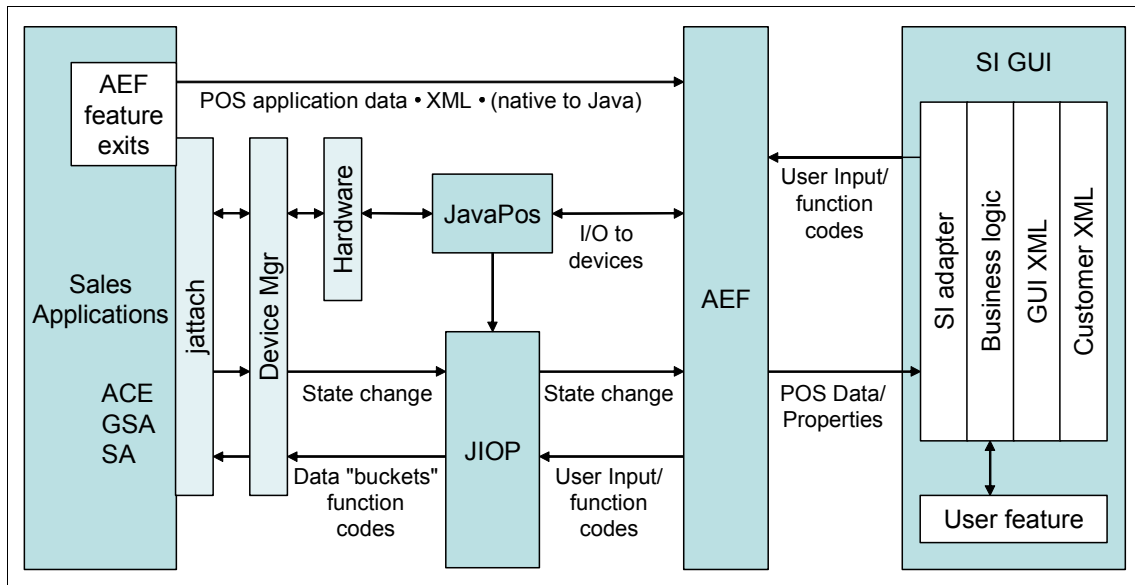


Figure 6-6 POS, JIOP, AEF, and store integrator GUI in a real terminal

► IBM WebSphere Enterprise Service Bus

Using the IBM SOA logical model, Easy Hogar y Construccin can define the following boundaries for the project scope:

- Business logic (business services) are outside WebSphere Enterprise Service Bus. WebSphere Enterprise Service Bus contains only integration logic or connectivity logic.
- Security and Management are loosely coupled to WebSphere Enterprise Service Bus. The policy decision point is outside WebSphere Enterprise Service Bus, and WebSphere Enterprise Service Bus is policy enforcement point.

- Service Registry is tightly coupled to WebSphere Enterprise Service Bus because the registry is a policy decision point for WebSphere Enterprise Service Bus and because WebSphere Enterprise Service Bus is a policy enforcement point for the registry.

Figure 6-7 shows the SOA logic model interaction with WebSphere Enterprise Service Bus and registry components.

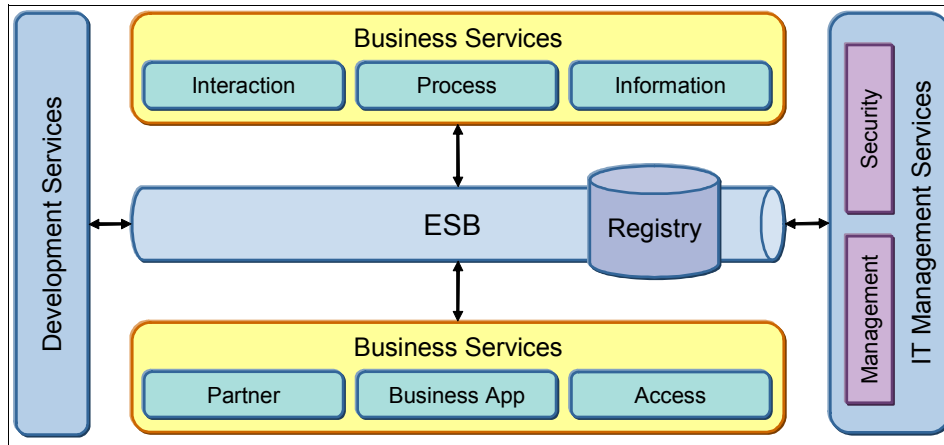


Figure 6-7 SOA logic model

► IBM WebSphere Service Registry and Repository

WebSphere Service Registry and Repository is the master metadata repository for service interaction that Provides Service Visibility and Governance for SOA services and policies.

Interactions between components

WebSphere Enterprise Service Bus and WebSphere Service Registry and Repository provide Easy Hogar y Construcccion with an infrastructure to support the SOA catalog as well as business and IT events.

When POS needs SOA interaction, Easy Hogar y Construcccion modifies the store integrator GUI to have access to transaction information. Thus, the company can retrieve any Web services through the ESB according to WebSphere Service Registry and Repository policies. Used with AEF, Easy Hogar y Construcccion can modify the GUI to add new multimedia or Web page content.

Figure 6-8 shows a sample for the operator GUI. Each button palette can invoke a local function or a remote service to complete transactions.

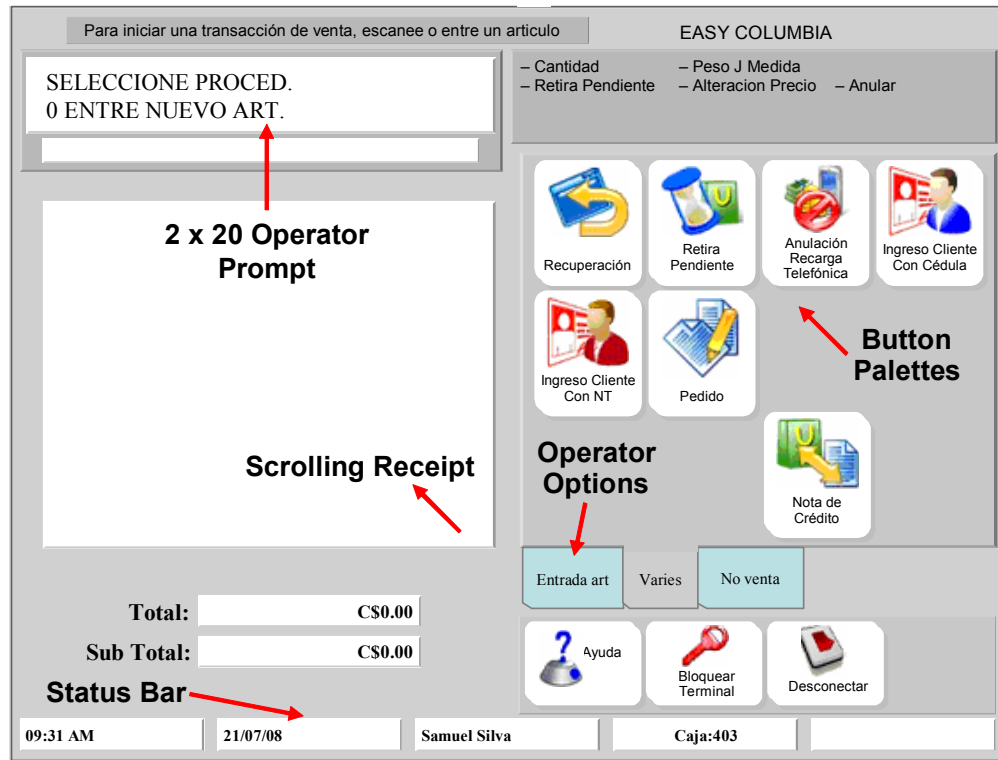


Figure 6-8 Store integrator UI operator window

Each Easy Hogar y Construcción application manages a set of services that can be composed to complete a business transaction.

Figure 6-9 describes the ESB mediation flows.

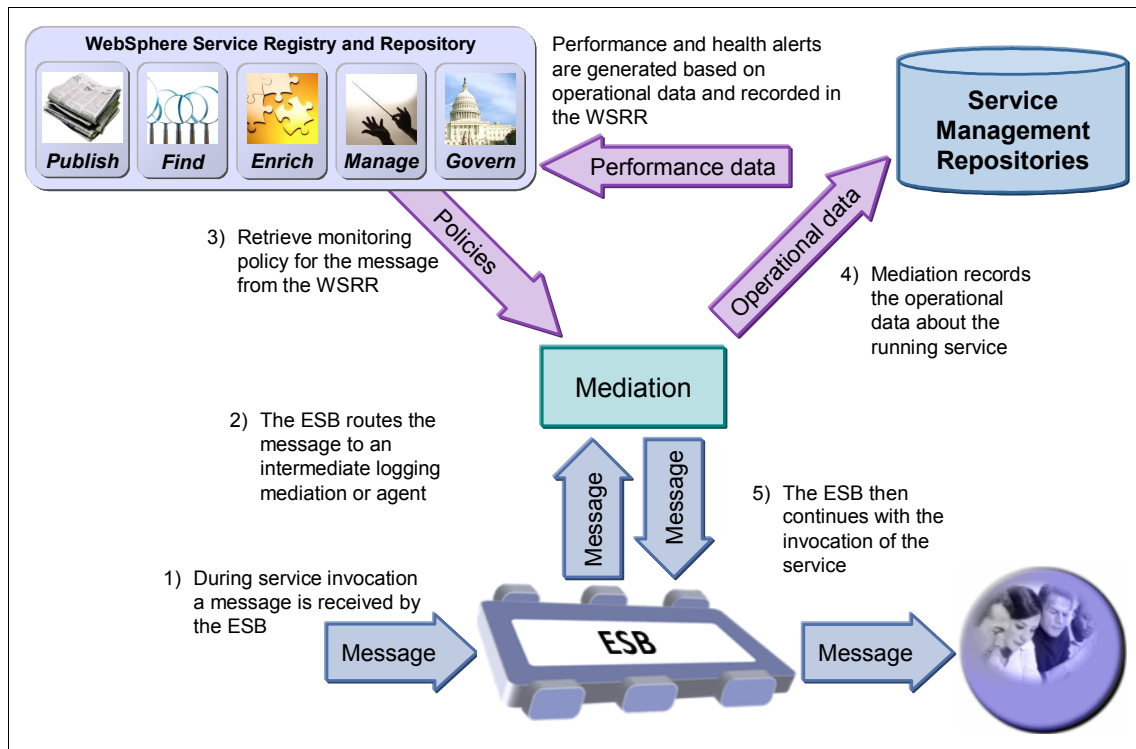


Figure 6-9 Easy Hogar y Construcción service usage

The Easy Hogar y Construcción architecture includes the following ESB main functions:

- Provide interaction patterns for communication protocols:
 - Supply basic connectivity to requesters and providers
 - Manage impact QoS (for example, reliable delivery and transactions)
 - Supply inherent interaction patterns (for example, request or reply, one-way, and pub or sub)
 - Provide support for the following standards:
 - Hypertext Transfer Protocol (HTTP), including SOAP/HTTP, and XML/HTTP
 - MQ (SOAP/JMS/MQ, XML/MQ, and text/MQ)
 - Adapters (existing, EIS)
 - WS-I, WS-Security

- ▶ Provide support for meta models and message models:
 - Describe messages that are exchanged with requesters and providers
 - Manage customized XML meta models (IXRetail)
 - Support multiple message content models
 - Support industry-standard models as well as enterprise specific models such as ARTS
- ▶ Provide support for mediation flows and manage service virtualization=:
 - Identity through routing basic, composed mediation patterns
 - Interaction (protocol or and pattern) through conversion
 - Interface through transformation
 - Aspect-oriented connectivity as security management, logging, and auditing

Figure 6-10 shows ESB Easy Hogar y Construcción extended scope that supports ARTS by the Meta models within the ESB.

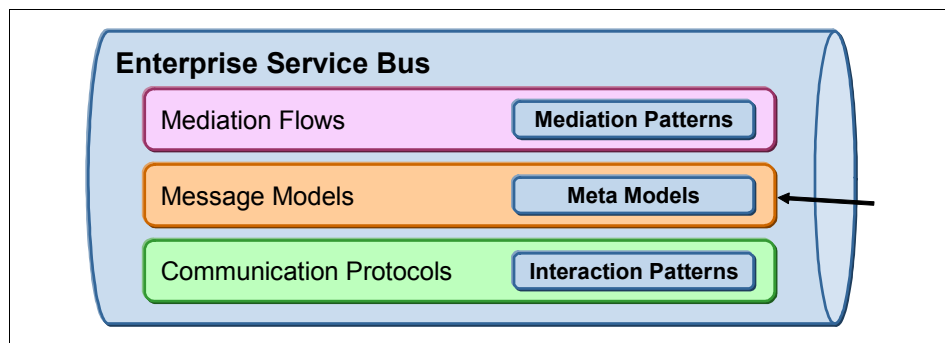


Figure 6-10 ESB patterns

Main challenges

The main challenges for the SOA store include:

- ▶ Understanding cross-industry best practices and business context for retail.
- ▶ Removing barriers to integration with back-end systems and sales channels.
- ▶ Enabling real-time event notification to be able to react to business issues.

6.1.6 Phase 2: Building a cross-channel strategy

Retail as a distributed business depends on the flow of information to and from the remainder of the enterprise as well as various external organizations. Easy

Hogar y Construcción needs a single platform to connect its business to its shoppers and to the business partners in its supply chain.

IBM WebSphere Commerce is the key component to managing a cross channel strategy using composite applications that integrate actual services catalog using the SOA logical model that we described in Figure 6-3 on page 428. WebSphere Commerce provides the service components shown in this figure, such as promotions, order, catalog, and so forth.

Cross-channel logical components

Figure 6-11 shows the logical areas of a cross channel architecture.

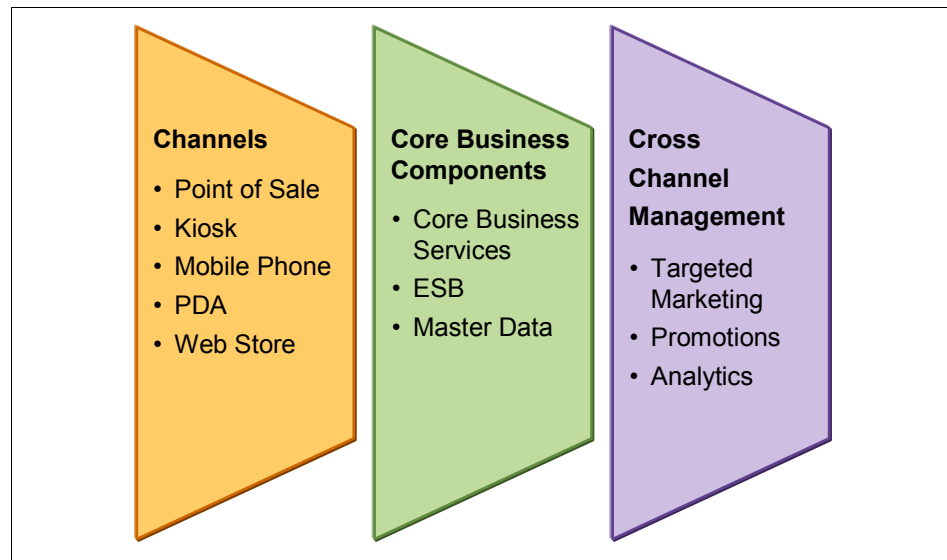


Figure 6-11 Logical model for a cross channel architecture

This cross-channel architecture includes the following components:

- ▶ *Channels* define, from a shopper's point of view, how a shopper can reach Easy Hogar y Construcción company and how the shopper sees the entire company as a unique brand, regardless of channels and external providers in the supply chain. Channels are about the shopping experience.
- ▶ *Core Business Components* describe, from the IT point of view, the main SOA infrastructure. This component is about governance, service catalog, and mediation.
- ▶ *Cross-channel Management* defines from a business point of view how Easy Hogar y Construcción can provide service rules that govern customer experience. Easy Hogar y Construcción knows that a cross channel strategy

must provide the necessary incentive to stimulate purchase behavior, to build the necessary awareness for the offer, and to develop optimum value perception. So, this logical level is business-oriented to produce and manage cross channel initiatives that avoid barriers to promotion participation.

Cross-channel software components

The main components of the cross-channel solution are:

- ▶ IBM WebSphere Commerce
- ▶ The SOA store model

These components are combined to create a unified shopping experience throughout all channels.

Main challenges

The cross channel strategy includes the following main challenges:

- ▶ Building a single view of the shopper from multiple channels, such as a Web site, kiosk, mobile device, POS, and call center.
- ▶ Providing the shopper with a single view of Easy Hogar y Construcción throughout multiple channels.
- ▶ Empowering the customer.
- ▶ Linking customer interactions with ongoing promotions.
- ▶ Ensuring that product information is relevant and accurate.
- ▶ Bringing differentiated services and programs to market more rapidly and efficiently.
- ▶ Gaining insight into the shopper's needs, wants, and value drivers.

6.1.7 Sample scenarios using the architecture

Easy Hogar y Construcción is using main basic business services with the ARTS SOA blueprint model. Each transaction, regardless of the channel, is recorded within the ARTS data model, allowing any other channel that might need this information to retrieve the transaction.

The ARTS data model contains special fields for logging transaction metadata. For example, this metadata can be used to trace the life cycle of an order that was purchased on the Web and subsequently returned in store. Using service components, any device is able to retrieve order payment information at store.

Process return case scenario

In this section, we expand on the purchase made on the Web. Now the shopper wants to return the product at a physical location. To complete this scenario, the following steps occur:

1. The shopper provides proof of purchase to the operator. If the proof of purchase is not available, the operator can retrieve purchase history using the shopper's identification and information.
2. The POS system retrieves the order information from the WebSphere Commerce order service component through WebSphere Enterprise Service Bus. WebSphere Commerce returns the order information to POS system.
3. The order is displayed at the POS system using the store integrator GUI, and the operator chooses which item the shopper is returning.
4. The operator must choose a return reason. If the reason indicates a fault of Easy Hogar y Construcción, such as a quality issue or damage to the merchandise, then a trigger is activated to present the shopper with an apology and to provide the shopper with a coupon for the next purchase.
5. The POS system processes the refund.
6. The transaction is saved on the main ARTS repository, which triggers an inventory service invocation to adjust the inventory.

Finish a Web or mobile transaction case scenario

This section describes a buy online, pick up in store scenario (referred to as BOPIS in this book). The shopper submitted the order through the Web and now wants to pick up the purchase from the store.

Note: IBM ACE POS can be running inside an IBM POS system, or it can be running in a special device such as a PDA, kiosk, or PC using AEF, depending on the store strategy.

To finish a Web or mobile transaction:

1. The shopper provides an identification to retrieve order information.
2. The POS system sends information using the ARTS XML schema through WebSphere Enterprise Service Bus to WebSphere Commerce. This message can be enhanced by the ESB mediation module (for example if additional data from other sources is needed).

3. The operator can add more items to the order if necessary. Using the touch screen monitor, Easy Hogar y Construcción can show advertisements that are tailored to this particular shopper.
4. The transaction is saved on the main ARTS repository, which triggers an inventory service invocation to adjust the inventory.

Figure 6-12 shows a high-level overview of the architecture components.

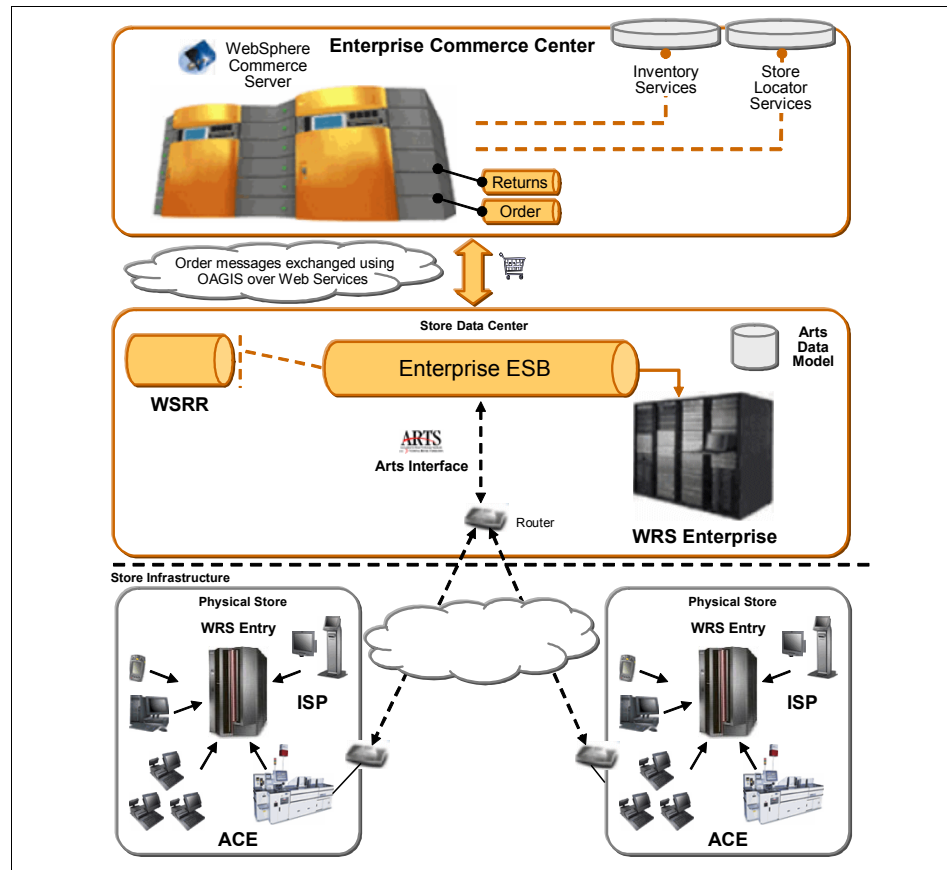


Figure 6-12 Solution high-level overview

Future enhancements

Working with this retail model, Easy Hogar y Construcción is ready to move to the next step:

- ▶ Add a business process engine such as IBM WebSphere Process Server to provide more complex cross channel and cross brand visibility into inventory, order status, and fulfillment activities.
- ▶ Build a business-to-business marketplace with value added networks.

6.1.8 Conclusion

The story of Easy Hogar y Construcción was successful because the company went from a small implementation to a large scale enterprise. Depending on a company's cross channel operational maturity and desired future state, a given organization must look for daily projects that lets them align their business and IT.

6.2 Targeted marketing and promotion for Web and mobile channel users

Easy Hogar y Construcción has multiple customer touchpoints that allow for convenient access to company offerings. The company is committed to providing service for its shoppers and a unique shopping experience. The marketing team is seeking new ideas to exploit the full potential of the current IT infrastructure.

6.2.1 Case scenario goals

Easy Hogar y Construcción needs to provide a consistent cross-channel experience where shoppers can initiate an order by mobile device, kiosk, Web, or POS and then modify, cancel, or submit the order using another channel when needed.

Easy Hogar y Construcción has developed the following initiatives:

- ▶ Attract shoppers to the store where they find a unique shopping experience as well as special in-store discounts.
- ▶ Offer store-only coupons and incentives online.
- ▶ Facilitate in-store pickup of online orders.
- ▶ Include easy views of available in-store inventory.
- ▶ Allow shoppers to find stores close to home.

Easy Hogar y Construcccion offers the following devices at the store:

- ▶ Web-enabled and SOA enabled POS systems that accept online orders and reservations
- ▶ Wireless kiosks where shoppers can place orders upon arrival at the store
- ▶ PDA to provide shoppers with a different purchase experience
- ▶ Wireless access for customer mobile devices

6.2.2 Easy Hogar y Construcccion registration sample

Using the cross channel software components that we described in 6.1, “Sample client solution” on page 422, Easy Hogar y Construcccion implements a marketing strategy for customer registration. Table 6-1 shows an overview of the business requirements for this sample.

Table 6-1 Cross-channel registration sample

Business Requirements	Strategy
<p>Easy Hogar y Construcccion needs to provide a unique cross channel customer registration procedure that lets the company work customer segmentation, targeted marketing using Web page, kiosk, or mobile device.</p> <p>Easy Hogar y Construcccion needs to manage alliances with other companies to let customers build home improvement tasks with the company.</p> <p>Easy Hogar y Construcccion needs to track daily registrations and needs to analyze data to generate new sales.</p>	<p>Registration provides customers a free insurance policy as a gift, free courses, and promotions that collect a customer's occupation.</p> <p>Each third-party alliance manages different targeted marketing promotions, and their mobile phones are the first tool to do marketing promotions.</p>

To fulfill the business requirements listed in Table 6-1 on page 442, Easy Hogar y Construcccion needs to implement the following steps:

1. Customize the Madisons Mobile Starter Store registration page.
2. Customize the Madisons Starter Store Web page, and include tags for analytics and e-Marketing Spots for promotions.
3. Load the third-party customer database.
4. Modify the store GUI to add a link to the Madisons Starter Store Web.
5. Modify the store GUI to run advertisements for Easy Hogar y Construcccion registration.

6. Modify the kiosk GUI to include access to the Madisons Starter Store Web.
7. Modify the kiosk GUI to include advertisements for Easy Hogar y Construcción registration.

Figure 6-13 shows a customer using kiosk solution to register at Easy Hogar y Construcción. The kiosk shows a registration option that is linked with WebSphere Commerce to apply e-Marketing Spots to qualifying customers.



Figure 6-13 Easy Hogar y Construcción Customer registration kiosk

6.2.3 Easy Hogar y Construcccion “do it yourself” project sample

Using the cross channel software components that we described in 6.1, “Sample client solution” on page 422, Easy Hogar y Construcccion implements a marketing strategy for shopper “do it yourself projects” (or *DIY projects*). Table 6-2 shows an overview of the business requirements for this sample.

Table 6-2 Cross-channel customer DIY project sample

Business Requirements	Strategy
Easy Hogar y Construcccion needs to provide shoppers a unique and convenient shopping experience at the store.	Shoppers can choose products for the project using the Easy Hogar y Construcccion PDA. They can place an order and can choose a sales assistant to help if needed, while walking in the store.
Easy Hogar y Construcccion provides convenient access to project templates to help customers choose the right products for their need.	Easy Hogar y Construcccion configures project templates at the Madisons Mobile Starter Store to let the shopper use as initial proposals. For example, Easy Hogar y Construcccion can provide templates for kitchen improvement.
Easy Hogar y Construcccion offers in-store wireless kiosks and PDA for shoppers.	Easy Hogar y Construcccion provides Wireless Order Entry. Shoppers can manage convenient access with a digital personal assistant or wireless access using a mobile device. These devices allow shoppers to order from a customized shopping list.
Easy Hogar y Construcccion needs to track project templates history and effectiveness.	<p>Suggest products based on previous selections to increase sales.</p> <p>Easy Hogar y Construcccion configures Targeted Marketing and promotions for Web and mobile channel for these shoppers.</p>

To fulfill the business requirements listed in Table 6-2 on page 444, the Easy Hogar y Construcción company needs to complete the following steps:

1. Customize the Madisons Mobile Starter Store order process.

Figure 6-14 shows a shopper using a mobile device to retrieve a special order that was made at home. The shopper retrieves the order to add new products and to verify product details.



Figure 6-14 In store Madisons Mobile Starter Store access

2. Customize the Madisons Starter Store Web page to include tags for analytics.
3. Configure targeted marketing and promotions for special projects.
4. Modify the ESB mediation to manage order management, which manages basic operations such as retrieve, modify, and cancel an order in WebSphere Commerce.
5. Modify the store GUI to add Web service call to order management.
6. Modify the store GUI to run advertisements for order management on Easy Hogar y Construcción channels.
7. Modify the kiosk GUI to include Web service call to order management.
8. Modify the kiosk GUI to include advertisements for Easy Hogar y Construcción order management.

Figure 6-15 shows a shopper's interaction with a sales assistant who manages a DIY project using a PDA device on a cross channel solution.



Figure 6-15 PDA wireless customer access

Figure 6-16 shows a POS solution that is integrated into a cross channel solution. In the picture, a shopper watches advertisements at a touch screen, flat panel while an operator uses a touch screen monitor to manage transaction operations. As explained previously, this solution retrieves order information from WebSphere Commerce.

The POS function provides a unique shopping experience and provides the company with feedback regarding consumer behavior across all channels. POS systems are always retrieving online information about promotions and product details.



Figure 6-16 Store GUI modification to retrieve order information

6.2.4 Easy Hogar y Construcccion family connections strategy sample

Using the cross channel software components that we described in 6.1, “Sample client solution” on page 422, Easy Hogar y Construcccion manages a marketing strategy for family connections. Table 6-3 lists the business requirements for this sample.

Table 6-3 Cross-channel family connections sample

Business Requirements	Strategy
<p>FAMILY CONNECTIONS</p> <p>Easy Hogar y Construcccion needs to offer a different kind of service keeping customers connected with their family and friends during important life events.</p> <p>Easy Hogar y Construcccion recognizes today's potential value for customer retention or customer acquisition such as young couples that are being invited to build projects with Easy Hogar y Construcccion.</p>	<p>Easy Hogar y Construcccion offers flower gifts to shoppers, where the shopper can search, buy, and pay using the same channels. After shoppers determine the template for the flower gift, the system must include an alert with the order, as part of a targeted marketing strategy.</p> <p>Gift Registry</p> <p>Social Commerce</p>
<p>SOCIAL COMMERCE</p> <p>Easy Hogar y Construcccion needs to offer access to a Web community where people can find professional services for the products that they buy. Shoppers can share product and service reviews, collaborating in a new social environment.</p> <p>Use online customer rating and feedback to improve shopper product selection and comparison shopping.</p>	<p>As part of the WebSphere Commerce evolution, Easy Hogar y Construcccion implements social commerce features.</p>

To fulfill the business requirements listed in Table 6-3 on page 448, the Easy Hogar y Construcccion company needs to implement the following steps:

1. Install the gift registry installation.
2. Install the sales center.
3. Install the social commerce feature.
4. Customize the Madisons Mobile Starter Store Registration page.
5. Customize the Madisons Starter Store Web page to include tags for analytics and tags marketing spots for promotions.

6. Modify the ESB mediation to manage order management.
7. Modify the store GUI to add Web service call to order management.
8. Modify the store GUI to run advertisements for order management on Easy Hogar y Construcción channels.
9. Modify the kiosk GUI to include Web service call to order management.
10. Modify the kiosk GUI to include advertisements for Easy Hogar y Construcción order management.

Figure 6-17 shows a shopper using the flower gift center to send flowers to family and friends on special dates. Easy Hogar y Construcción uses this service to reward shoppers based on segmentation and purchase history. This kiosk strategy is also used with gift registry to support an entire loyalty program for shoppers.



Figure 6-17 Kiosk using Family Connections Strategy

6.3 Cross-channel precision marketing

To maximize conversion rates in a cross channel environment, Easy Hogar y Construcción wants to track and analyze cross channel customer behavior to gain insights and then apply those insights to its sales and marketing initiatives.

6.3.1 Marketing strategy

Easy Hogar y Construcción develops a cross-channel plan that includes the following marketing and operational goals:

- ▶ Create product offers and services for channels.
- ▶ Coordinate brand look.
- ▶ Execute unified marketing plans between channels.
- ▶ Improve new programs for new channels such as mobile strategy.
- ▶ Create single brand identity across channels.
- ▶ Continuously improve operational excellence across channels.
- ▶ Determine new customers segments that might be worthwhile to target in the Web site, kiosk, or any other channel.

6.3.2 Indicators

Easy Hogar y Construcción needs to track orders by channel to develop new strategies. Table 6-4 shows some key indicators to measure.

Table 6-4 Cross-channel main indicators

Operations	Indicators
Purchase	<ul style="list-style-type: none">▶ Customer segmentation by channel▶ Average conversion rate▶ Average order value▶ Average per visit / channel value▶ Average ROI▶ Percentage revenue from new visitors▶ New customer on first interaction index▶ Shopping Discounts by Channel
Order Input	<ul style="list-style-type: none">▶ Percentage visits by channel▶ Percentage goal conversion by channel▶ Percentage visit by channel▶ Average ROI by channel▶ Cancelled Orders by Type, Reason

Operations	Indicators
Marketing	<ul style="list-style-type: none"> ▶ Customer Satisfaction by Channel ▶ Number of advertisements clicked ▶ Percentage brand engagement ▶ Goal conversion index by channel ▶ Percentage of new versus returning visitors ▶ Percentage of new versus returning customers

6.3.3 Using WebSphere Commerce features

Easy Hogar y Construcccion uses the marketing activities described in this section for planned customer interactions.

Campaigns

Using WebSphere Commerce campaigns for customers, Easy Hogar y Construcccion manages Web and e-mail activities:

- ▶ Campaign using customer occupation

Every 2 weeks there is a special event using shopper occupations (for example, painters, architects, or designers). This feature lets the company collect all professionals contact information to offer special prices for affiliated occupations and projects. If the shoppers bring their customers to Easy Hogar y Construcccion, the shoppers become strategic partners of a new social community.
- ▶ Campaign for third-party alliances

This campaign uses third-party databases from alliances that let the company offer special targeted promotions. For public services companies, Easy Hogar y Construcccion offers special events each week to promote “green home projects” that offer special devices that consume less energy. In this example, e-mail marketing allows Easy Hogar y Construcccion to reach shoppers who have not yet visited a store. Finally, e-mail and Short Message Service (SMS) marketing lets the company keep regular shoppers up-to-date regarding upcoming events and products.

Web activities

Web activities let Easy Hogar y Construcción control predefined e-Marketing Spots on the store pages. Table 6-5 describes these Web activities.

Table 6-5 Web activities samples

Pages	Goal
StoreHomePage e-Marketing Spot	Display ads about how to save money using less energy consumption devices
Shopping Cart e-Marketing Spot	Display based on purchase conditions how to get free shipping
Product Display Page e-Marketing Spot	Upsell merchandising association for the current catalog.

Dialog activities

Dialog activities let Easy Hogar y Construcción to look for methods to automate marketing actions based on specific customer behaviors over time. Using trigger, target, and action components, Easy Hogar y Construcción can generate the following new dialog activities:

- ▶ When a shopper places an order, check the past purchase history credit. If payment history is successful for the last 6 months, move the shopper to the *pre-approval building budget* customer segment.
- ▶ When shopper information comes from a third-party source, for example public service energy company, then offer the shopper green products to save energy at home for 2 months.
- ▶ When the shopper's age is between 20 and 30 and the shopper is married, move the shopper to the *young new couples* customer segment and send the shopper SMS about saving money for home projects. If the shopper has pets, then add the shopper to a special *pet* customer segment as well.

6.4 Summary

A cross channel strategy works with the complete picture to provide an understanding of how to create a new shopping experience that aligns business and IT. The key is to understand the process that really work and how to measure your success.

The successful implementation for Easy Hogar y Construcción had the main keys of IT and human factors. IT was supported by an excellent team of individuals that were hired to develop a new vision and to make a self-consulting process that was delivered later by IBM and its business partners. This team was able to develop a methodology that provide that the human factor is a critical asset when building a company vision for the future.



A

Samples of WebSphere Commerce SOA Service Request and Response

This appendix includes samples of WebSphere Commerce SOA Service Request and Response used by the distributed order management (referred to as *DOM* throughout this book) integration solution.

A.1 ProcessInventoryRequirement with action code ReserveInventory request

Example A-1 shows the ProcessInventoryRequirement request with the action code ReserveInventory request.

Example A-1 ProcessInventoryRequirement request with action code ReserveInventory

```
<_inv:ProcessInventoryRequirement
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:_inv="http://www.ibm.com/xmlns/prod/commerce/9/inventory"
xmlns:_ord="http://www.ibm.com/xmlns/prod/commerce/9/order"
xmlns:_wcf="http://www.ibm.com/xmlns/prod/commerce/9/foundation"
xmlns:oa="http://www.openapplications.org/oagis/9" releaseID="9.0"
versionID="6.0.0.4">
  <oa:ApplicationArea xsi:type="_wcf:ApplicationAreaType">
    <oa:CreationDateTime>2009-08-11T16:09:28.218Z</oa:CreationDateTime>
    <oa:BODID>58111cf0-8691-11de-845f-84214a818784</oa:BODID>
    <_wcf:BusinessContext/>
  </oa:ApplicationArea>
  <_inv:DataArea>
    <oa:Process>
      <oa:ActionCriteria>
        <oa:ActionExpression actionCode="ReserveInventory"
expressionLanguage="_wcf:XPath">/InventoryRequirement[1]</oa:ActionExpr
ession>
        </oa:ActionCriteria>
      </oa:Process>
    <_inv:InventoryRequirement shipAsComplete="true">
      <_ord:OrderIdentifier>
        <_wcf:UniqueID>12501</_wcf:UniqueID>
      </_ord:OrderIdentifier>
      <_ord:StoreIdentifier>
        <_wcf:UniqueID>10001</_wcf:UniqueID>
        <_wcf:ExternalIdentifier>
          <_wcf:NameIdentifier>Madisons</_wcf:NameIdentifier>
        </_wcf:ExternalIdentifier>
      </_ord:StoreIdentifier>
      <_ord:BuyerIdentifier>
        <_wcf:UniqueID>2</_wcf:UniqueID>
        <_wcf:DistinguishedName>uid=wangful,o=default
organization,o=root organization</_wcf:DistinguishedName>
      </_ord:BuyerIdentifier>
      <_ord:OrderAmount>
```

```

        <_wcf:GrandTotal currency="USD">323.99000</_wcf:GrandTotal>
        <_wcf:TotalProductPrice
currency="USD">449.99000</_wcf:TotalProductPrice>
        <_wcf:TotalAdjustment
currency="USD">-126.00000</_wcf:TotalAdjustment>
        <_wcf:Adjustment>
            <_wcf:Usage>Discount</_wcf:Usage>
            <_wcf:Code>Furniture Category Discount</_wcf:Code>
            <_wcf:Description language="-1">Save 20% on
Furniture!</_wcf:Description>
            <_wcf:Amount currency="USD">-90.00000</_wcf:Amount>
            <_wcf:DisplayLevel>OrderItem</_wcf:DisplayLevel>
        </_wcf:Adjustment>
        <_wcf:Adjustment>
            <_wcf:Usage>Discount</_wcf:Usage>
            <_wcf:Code>Save 10% on all orders today</_wcf:Code>
            <_wcf:Description language="-1">Save 10% on all
orders</_wcf:Description>
            <_wcf:Amount currency="USD">-36.00000</_wcf:Amount>
            <_wcf:DisplayLevel>Order</_wcf:DisplayLevel>
        </_wcf:Adjustment>
        <_wcf:TotalShippingCharge
currency="USD">0.00000</_wcf:TotalShippingCharge>
        <_wcf:TotalSalesTax currency="USD">0.00000</_wcf:TotalSalesTax>
        <_wcf:TotalShippingTax
currency="USD">0.00000</_wcf:TotalShippingTax>
    </_ord:OrderAmount>
    <_ord:OrderPaymentInfo/>
    <_ord:OrderStatus prepareIndicator="false">
        <_ord:Status>P</_ord:Status>
    </_ord:OrderStatus>

    <_ord:LastUpdateDate>2009-08-11T16:08:52.687Z</_ord:LastUpdateDate>
    <_ord:OrderItem>
        <_ord:OrderItemIdentifier>
            <_wcf:UniqueID>45001</_wcf:UniqueID>
        </_ord:OrderItemIdentifier>
        <_ord:CatalogEntryIdentifier>
            <_wcf:UniqueID>10002</_wcf:UniqueID>
            <_wcf:ExternalIdentifier>
                <_wcf:PartNumber>FUL0-0101</_wcf:PartNumber>
            </_wcf:ExternalIdentifier>
        </_ord:CatalogEntryIdentifier>
        <_ord:Quantity uom="C62">1.0</_ord:Quantity>
        <_ord:ContractIdentifier>

```

```

        <_wcf:UniqueID>10001</_wcf:UniqueID>
    </_ord:ContractIdentifier>
    <_ord:OfferID>10002</_ord:OfferID>
    <_ord:OrderItemShippingInfo expedite="false">
        <_ord:ShippingAddress>
            <_wcf:ContactInfoIdentifier>
                <_wcf:UniqueID>10076</_wcf:UniqueID>
                <_wcf:ExternalIdentifier>
                    <_wcf:ContactInfoNickName>Calgary
Mall</_wcf:ContactInfoNickName>
                </_wcf:ExternalIdentifier>
            </_wcf:ContactInfoIdentifier>
            <_wcf:ContactName>
                <_wcf:PersonTitle></_wcf:PersonTitle>
                <_wcf:LastName>Calgary Mall</_wcf:LastName>
                <_wcf:FirstName></_wcf:FirstName>
                <_wcf:MiddleName></_wcf:MiddleName>
            </_wcf:ContactName>
            <_wcf:Address>
                <_wcf:AddressLine>1025 Cameron Ave SW</_wcf:AddressLine>
                <_wcf:AddressLine></_wcf:AddressLine>
                <_wcf:AddressLine></_wcf:AddressLine>
                <_wcf:City> Calgary</_wcf:City>

        <_wcf:StateOrProvinceName>Alberta</_wcf:StateOrProvinceName>
            <_wcf:Country>Canada</_wcf:Country>
            <_wcf:PostalCode>T2T 0K4</_wcf:PostalCode>
        </_wcf:Address>
        <_wcf:Telephone1 publish="true">
            <_wcf:Value></_wcf:Value>
        </_wcf:Telephone1>
        <_wcf:Telephone2 publish="true">
            <_wcf:Value></_wcf:Value>
        </_wcf:Telephone2>
        <_wcf:EmailAddress1>
            <_wcf:Value>admin@madisons.ca</_wcf:Value>
        </_wcf:EmailAddress1>
        <_wcf:EmailAddress2>
            <_wcf:Value></_wcf:Value>
        </_wcf:EmailAddress2>
        <_wcf:Fax1>
            <_wcf:Value></_wcf:Value>
        </_wcf:Fax1>
        <_wcf:Fax2>
            <_wcf:Value></_wcf:Value>
    </_ord:OrderItemShippingInfo>

```



```

        </_wcf:Fax2>
    </_ord:ShippingAddress>
    <_ord:ShippingMode>
        <_ord:ShippingModeIdentifier>
            <_ord:UniqueID>10001</_ord:UniqueID>
            <_ord:ExternalIdentifier>
                <_ord:ShipModeCode>PickupInStore</_ord:ShipModeCode>
            </_ord:ExternalIdentifier>
        </_ord:ShippingModeIdentifier>
        <_ord:Description language="-1">Pickup in
store</_ord:Description>
    </_ord:ShippingMode>
    <_ord:PhysicalStoreIdentifier>
        <_wcf:UniqueID>10026</_wcf:UniqueID>
        <_wcf:ExternalIdentifier>Calgary
Mall</_wcf:ExternalIdentifier>
    </_ord:PhysicalStoreIdentifier>
</_ord:OrderItemShippingInfo>
<_ord:OrderItemStatus>
    <_ord:Status>P</_ord:Status>
    <_ord:InventoryStatus>Available</_ord:InventoryStatus>
    <_ord:FulfillmentStatus>Unreleased</_ord:FulfillmentStatus>
</_ord:OrderItemStatus>
<_ord:OrderItemFulfillmentInfo>

<_ord:AvailableDate>2009-08-11T16:08:54.609Z</_ord:AvailableDate>

<_ord:ExpectedShipDate>2009-08-11T16:09:27.921Z</_ord:ExpectedShipDate>
</_ord:OrderItemFulfillmentInfo>
<_ord:FulfillmentCenter>
    <_ord:FulfillmentCenterIdentifier>
        <_wcf:UniqueID>10076</_wcf:UniqueID>
        <_wcf:Name>Calgary Mall</_wcf:Name>
    </_ord:FulfillmentCenterIdentifier>
</_ord:FulfillmentCenter>
<_ord:CorrelationGroup>45001</_ord:CorrelationGroup>
<_ord:CreateDate>2009-08-11T15:36:06.781Z</_ord:CreateDate>

<_ord:LastUpdateDate>2009-08-11T16:08:52.812Z</_ord:LastUpdateDate>
    <_ord:UsableShippingChargePolicy>
        <_wcf:UniqueID>-7001</_wcf:UniqueID>
        <_wcf:ExternalIdentifier>
            <_wcf:Name>StandardShippingChargeBySeller</_wcf:Name>
            <_wcf:Type>ShippingCharge</_wcf:Type>
        </_wcf:ExternalIdentifier>
    </_ord:UsableShippingChargePolicy>

```

```

        </_ord:UsableShippingChargePolicy>
    </_ord:OrderItem>
</_inv:InventoryRequirement>
</_inv:DataArea>
</_inv:ProcessInventoryRequirement>

```

A.2 ProcessInventoryRequirement with action code ReserveInventory response

Example A-2 shows the ProcessInventoryRequirement request with the action code ReserveInventory response.

Example A-2 ProcessInventoryRequirement with action code ReserveInventory response

```

<_inv:AcknowledgeInventoryRequirement
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:_inv="http://www.ibm.com/xmlns/prod/commerce/9/inventory"
xmlns:_ord="http://www.ibm.com/xmlns/prod/commerce/9/order"
xmlns:_wcf="http://www.ibm.com/xmlns/prod/commerce/9/foundation"
xmlns:oa="http://www.openapplications.org/oagis/9" releaseID="9.0">
  <oa:ApplicationArea xsi:type="_wcf:ApplicationAreaType">
    <oa:CreationDateTime>2009-08-11T16:09:31.953Z</oa:CreationDateTime>
    <oa:BODID>5a489660-8691-11de-845f-84214a818784</oa:BODID>
  </oa:ApplicationArea>
  <_inv:DataArea>
    <oa:Acknowledge>
      <oa:OriginalApplicationArea>
        <oa:CreationDateTime>2009-08-11T16:09:28.218Z</oa:CreationDateTime>
        <oa:BODID>58111cf0-8691-11de-845f-84214a818784</oa:BODID>
      </oa:OriginalApplicationArea>
    </oa:Acknowledge>
    <_inv:InventoryRequirement shipAsComplete="true">
      <_ord:OrderIdentifier>
        <_wcf:UniqueID>12501</_wcf:UniqueID>
      </_ord:OrderIdentifier>
      <_ord:StoreIdentifier>
        <_wcf:UniqueID>10001</_wcf:UniqueID>
        <_wcf:ExternalIdentifier>
          <_wcf:NameIdentifier>Madisons</_wcf:NameIdentifier>
        </_wcf:ExternalIdentifier>
      </_ord:StoreIdentifier>
      <_ord:BuyerIdentifier>

```

```

    <_wcf:UniqueID>2</_wcf:UniqueID>
    <_wcf:DistinguishedName>uid=wangful,o=default organization,o=root
organization</_wcf:DistinguishedName>
  </_ord:BuyerIdentifier>
  <_ord:OrderAmount>
    <_wcf:GrandTotal currency="USD">323.99000</_wcf:GrandTotal>
    <_wcf:TotalProductPrice currency="USD">449.99000</_wcf:TotalProductPrice>
    <_wcf:TotalAdjustment currency="USD">-126.00000</_wcf:TotalAdjustment>
    <_wcf:Adjustment>
      <_wcf:Usage>Discount</_wcf:Usage>
      <_wcf:Code>Furniture Category Discount</_wcf:Code>
      <_wcf:Description language="-1">Save 20% on Furniture!</_wcf:Description>
      <_wcf:Amount currency="USD">-90.00000</_wcf:Amount>
      <_wcf:DisplayLevel>OrderItem</_wcf:DisplayLevel>
    </_wcf:Adjustment>
    <_wcf:Adjustment>
      <_wcf:Usage>Discount</_wcf:Usage>
      <_wcf:Code>Save 10% on all orders today</_wcf:Code>
      <_wcf:Description language="-1">Save 10% on all orders</_wcf:Description>
      <_wcf:Amount currency="USD">-36.00000</_wcf:Amount>
      <_wcf:DisplayLevel>Order</_wcf:DisplayLevel>
    </_wcf:Adjustment>
    <_wcf:TotalShippingCharge currency="USD">0.00000</_wcf:TotalShippingCharge>
    <_wcf:TotalSalesTax currency="USD">0.00000</_wcf:TotalSalesTax>
    <_wcf:TotalShippingTax currency="USD">0.00000</_wcf:TotalShippingTax>
  </_ord:OrderAmount>
  <_ord:OrderPaymentInfo/>
  <_ord:OrderStatus prepareIndicator="false">
    <_ord:Status>P</_ord:Status>
  </_ord:OrderStatus>
  <_ord:LastUpdateDate>2009-08-11T16:08:52.687Z</_ord:LastUpdateDate>
  <_ord:OrderItem>
    <_ord:OrderItemIdentifier>
      <_wcf:UniqueID>45001</_wcf:UniqueID>
    </_ord:OrderItemIdentifier>
    <_ord:CatalogEntryIdentifier>
      <_wcf:UniqueID>10002</_wcf:UniqueID>
      <_wcf:ExternalIdentifier>
        <_wcf:PartNumber>FUL0-0101</_wcf:PartNumber>
      </_wcf:ExternalIdentifier>
    </_ord:CatalogEntryIdentifier>
    <_ord:Quantity uom="C62">1.0</_ord:Quantity>
    <_ord:ContractIdentifier>
      <_wcf:UniqueID>10001</_wcf:UniqueID>
    </_ord:ContractIdentifier>
  </_ord:OrderItem>

```

```

<_ord:OfferID>10002</_ord:OfferID>
<_ord:OrderItemShippingInfo expedite="false">
  <_ord:ShippingAddress>
    <_wcf:ContactInfoIdentifier>
      <_wcf:UniqueID>10076</_wcf:UniqueID>
      <_wcf:ExternalIdentifier>
        <_wcf:ContactInfoNickName>Calgary Mall</_wcf:ContactInfoNickName>
      </_wcf:ExternalIdentifier>
    </_wcf:ContactInfoIdentifier>
    <_wcf:ContactName>
      <_wcf:PersonTitle></_wcf:PersonTitle>
      <_wcf:LastName>Calgary Mall</_wcf:LastName>
      <_wcf:FirstName></_wcf:FirstName>
      <_wcf:MiddleName></_wcf:MiddleName>
    </_wcf:ContactName>
    <_wcf:Address>
      <_wcf:AddressLine>1025 Cameron Ave SW</_wcf:AddressLine>
      <_wcf:AddressLine></_wcf:AddressLine>
      <_wcf:AddressLine></_wcf:AddressLine>
      <_wcf:City> Calgary</_wcf:City>
      <_wcf:StateOrProvinceName>Alberta</_wcf:StateOrProvinceName>
      <_wcf:Country>Canada</_wcf:Country>
      <_wcf:PostalCode>T2T 0K4</_wcf:PostalCode>
    </_wcf:Address>
    <_wcf:Telephone1 publish="true">
      <_wcf:Value></_wcf:Value>
    </_wcf:Telephone1>
    <_wcf:Telephone2 publish="true">
      <_wcf:Value></_wcf:Value>
    </_wcf:Telephone2>
    <_wcf:EmailAddress1>
      <_wcf:Value>admin@madisons.ca</_wcf:Value>
    </_wcf:EmailAddress1>
    <_wcf:EmailAddress2>
      <_wcf:Value></_wcf:Value>
    </_wcf:EmailAddress2>
    <_wcf:Fax1>
      <_wcf:Value></_wcf:Value>
    </_wcf:Fax1>
    <_wcf:Fax2>
      <_wcf:Value></_wcf:Value>
    </_wcf:Fax2>
  </_ord:ShippingAddress>
  <_ord:ShippingMode>
    <_ord:ShippingModeIdentifier>

```

```

        <_ord:UniqueID>10001</_ord:UniqueID>
        <_ord:ExternalIdentifier>
            <_ord:ShipModeCode>PickupInStore</_ord:ShipModeCode>
        </_ord:ExternalIdentifier>
    </_ord:ShippingModeIdentifier>
    <_ord:Description language="-1">Pickup in store</_ord:Description>
</_ord:ShippingMode>
    <_ord:PhysicalStoreIdentifier>
        <_wcf:UniqueID>10026</_wcf:UniqueID>
        <_wcf:ExternalIdentifier>Calgary Mall</_wcf:ExternalIdentifier>
    </_ord:PhysicalStoreIdentifier>
</_ord:OrderItemShippingInfo>
    <_ord:OrderItemStatus>
        <_ord:Status>P</_ord:Status>
        <_ord:InventoryStatus>Allocated</_ord:InventoryStatus>
        <_ord:FulfillmentStatus>Unreleased</_ord:FulfillmentStatus>
    </_ord:OrderItemStatus>
    <_ord:OrderItemFulfillmentInfo>
        <_ord:AvailableDate>2009-08-11T16:09:31.468Z</_ord:AvailableDate>
        <_ord:ExpectedShipDate>2009-08-11T16:09:27.921Z</_ord:ExpectedShipDate>
    </_ord:OrderItemFulfillmentInfo>
    <_ord:FulfillmentCenter>
        <_ord:FulfillmentCenterIdentifier>
            <_wcf:UniqueID>10076</_wcf:UniqueID>
            <_wcf:Name>Calgary Mall</_wcf:Name>
        </_ord:FulfillmentCenterIdentifier>
    </_ord:FulfillmentCenter>
    <_ord:CorrelationGroup>45001</_ord:CorrelationGroup>
    <_ord:CreateDate>2009-08-11T15:36:06.781Z</_ord:CreateDate>
    <_ord:LastUpdateDate>2009-08-11T16:08:52.812Z</_ord:LastUpdateDate>
    <_ord:UsableShippingChargePolicy>
        <_wcf:UniqueID>-7001</_wcf:UniqueID>
        <_wcf:ExternalIdentifier>
            <_wcf:Name>StandardShippingChargeBySeller</_wcf:Name>
            <_wcf:Type>ShippingCharge</_wcf:Type>
        </_wcf:ExternalIdentifier>
    </_ord:UsableShippingChargePolicy>
</_ord:OrderItem>
</_inv:InventoryRequirement>
</_inv:DataArea>
</_inv:AcknowledgeInventoryRequirement>

```

A.3 ProcessInventory with action code CancelInventoryReservation request

Example A-3 shows the ProcessInventoryRequirement with the action code CancelInventoryReservation request.

Example A-3 ProcessInventoryRequirement with action code CancelInventoryReservation request

```
<_inv:ProcessInventoryRequirement releaseID="9.0" versionID="6.0.0.4"
xmlns:_inv="http://www.ibm.com/xmlns/prod/commerce/9/inventory"
xmlns:_ord="http://www.ibm.com/xmlns/prod/commerce/9/order"
xmlns:_wcf="http://www.ibm.com/xmlns/prod/commerce/9/foundation"
xmlns:oa="http://www.openapplications.org/oagis/9"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <oa:ApplicationArea xsi:type="_wcf:ApplicationAreaType">
    <oa:CreationDateTime>2009-04-02T08:28:47.718Z</oa:CreationDateTime>
    <oa:BODID>4929ef20-1f60-11de-8f9a-834749d4773b</oa:BODID>
    <_wcf:BusinessContext>
      <_wcf:ContextData name="langId">-1</_wcf:ContextData>
      <_wcf:ContextData name="storeId">553</_wcf:ContextData>
    </_wcf:BusinessContext>
  </oa:ApplicationArea>
  <_inv:DataArea>
    <oa:Process>
      <oa:ActionCriteria>
        <oa:ActionExpression actionCode="CancelInventoryReservation"
expressionLanguage="_wcf:XPath">/InventoryRequirement[1]</oa:ActionExpression>
        </oa:ActionCriteria>
      </oa:Process>
    <_inv:InventoryRequirement>
      <_ord:OrderIdentifier>
        <_wcf:UniqueID>14507</_wcf:UniqueID>
      </_ord:OrderIdentifier>
      <_ord:StoreIdentifier>
        <_wcf:UniqueID>553</_wcf:UniqueID>
      </_ord:StoreIdentifier>
      <_ord:BuyerIdentifier>
        <_wcf:UniqueID>100000000501</_wcf:UniqueID>
        <_wcf:DistinguishedName>uid=user,ou=extended sites seller organizationsample
b2c store,o=extended sites seller organization,o=root
organization</_wcf:DistinguishedName>
      </_ord:BuyerIdentifier>
      <_ord:OrderStatus prepareIndicator="false">
        <_ord:Status>P</_ord:Status>
      </_ord:OrderStatus>
    </_inv:InventoryRequirement>
  </_inv:DataArea>
</_inv:ProcessInventoryRequirement>
```

```

</_ord:OrderStatus>
<_ord:LastUpdateDate>2009-01-13T09:57:33.420Z</_ord:LastUpdateDate>
<_ord:PlacedDate>2009-01-13T09:57:33.420Z</_ord:PlacedDate>
<_ord:OrderItem>
  <_ord:OrderItemIdentifier>
    <_wcf:UniqueID>10045007</_wcf:UniqueID>
  </_ord:OrderItemIdentifier>
  <_ord:CatalogEntryIdentifier>
    <_wcf:UniqueID>55301088100</_wcf:UniqueID>
  </_ord:CatalogEntryIdentifier>
  <_ord:Quantity>2.0</_ord:Quantity>
  <_ord:ContractIdentifier>
    <_wcf:UniqueID>50300000003</_wcf:UniqueID>
  </_ord:ContractIdentifier>
  <_ord:OfferID>55301088100</_ord:OfferID>
  <_ord:OrderItemAmount freeGift="false">
    <_wcf:UnitPrice>
      <_wcf:Price currency="USD">55.00000</_wcf:Price>
      <_wcf:Quantity uom="C62">1.0</_wcf:Quantity>
    </_wcf:UnitPrice>
    <_wcf:OrderItemPrice currency="USD">110.00000</_wcf:OrderItemPrice>
    <_wcf:ShippingCharge currency="USD">0.00000</_wcf:ShippingCharge>
    <_wcf:SalesTax currency="USD">0.00000</_wcf:SalesTax>
    <_wcf:ShippingTax currency="USD">0.00000</_wcf:ShippingTax>
  </_ord:OrderItemAmount>
  <_ord:OrderItemShippingInfo expedite="false">
    <_ord:ShippingAddress>
      <_wcf:ContactInfoIdentifier>
        <_wcf:UniqueID>100000000051</_wcf:UniqueID>
        <_wcf:ExternalIdentifier>
          <_wcf:ContactInfoNickName>user</_wcf:ContactInfoNickName>
        </_wcf:ExternalIdentifier>
      </_wcf:ContactInfoIdentifier>
      <_wcf:ContactName>
        <_wcf:PersonTitle/>
        <_wcf:LastName>user</_wcf:LastName>
        <_wcf:FirstName/>
        <_wcf:MiddleName/>
      </_wcf:ContactName>
      <_wcf:Address>
        <_wcf:AddressLine>as</_wcf:AddressLine>
        <_wcf:AddressLine/>
        <_wcf:AddressLine/>
        <_wcf:City>sa</_wcf:City>
        <_wcf:StateOrProvinceName>aa</_wcf:StateOrProvinceName>
      </_wcf:Address>
    </_ord:ShippingAddress>
  </_ord:OrderItemShippingInfo>
</_ord:OrderItem>

```

```

        <_wcf:Country>AF</_wcf:Country>
        <_wcf:PostalCode>ss</_wcf:PostalCode>
    </_wcf:Address>
    <_wcf:Telephone1 publish="true">
        <_wcf:Value/>
    </_wcf:Telephone1>
    <_wcf:Telephone2 publish="true">
        <_wcf:Value/>
    </_wcf:Telephone2>
    <_wcf:EmailAddress1>
        <_wcf:Value/>
    </_wcf:EmailAddress1>
    <_wcf:EmailAddress2>
        <_wcf:Value/>
    </_wcf:EmailAddress2>
    <_wcf:Fax1>
        <_wcf:Value/>
    </_wcf:Fax1>
    <_wcf:Fax2>
        <_wcf:Value/>
    </_wcf:Fax2>
</_ord:ShippingAddress>
<_ord:ShippingMode>
    <_ord:ShippingModeIdentifier>
        <_ord:UniqueID>5030051</_ord:UniqueID>
    </_ord:ShippingModeIdentifier>
    <_ord:Description language="-1">Mail</_ord:Description>
</_ord:ShippingMode>
</_ord:OrderItemShippingInfo>
<_ord:OrderItemStatus>
    <_ord:Status>M</_ord:Status>
    <_ord:InventoryStatus>Allocated</_ord:InventoryStatus>
    <_ord:FulfillmentStatus>Unreleased</_ord:FulfillmentStatus>
</_ord:OrderItemStatus>
<_ord:OrderItemFulfillmentInfo>
    <_ord:ExpectedShipDate>2009-01-13T09:57:54.578Z</_ord:ExpectedShipDate>
</_ord:OrderItemFulfillmentInfo>
<_ord:FulfillmentCenter>
    <_ord:FulfillmentCenterIdentifier>
        <_wcf:UniqueID/>
    </_ord:FulfillmentCenterIdentifier>
</_ord:FulfillmentCenter>
</_ord:OrderItem>
</_inv:InventoryRequirement>
</_inv:DataArea>
</_inv:ProcessInventoryRequirement>

```

A.4 ProcessInventoryRequirement with action code CancelInventoryReservation response

Example A-4 shows the ProcessInventoryRequirement with the action code CancelInventoryReservation response.

Example A-4 ProcessInventoryRequirement with action code CancelInventoryReservation response

```
<_inv:AcknowledgeInventoryRequirement releaseID="9.0"
xmlns:_inv="http://www.ibm.com/xmlns/prod/commerce/9/inventory"
xmlns:_ord="http://www.ibm.com/xmlns/prod/commerce/9/order"
xmlns:_wcf="http://www.ibm.com/xmlns/prod/commerce/9/foundation"
xmlns:oa="http://www.openapplications.org/oagis/9"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <oa:ApplicationArea xsi:type="_wcf:ApplicationAreaType">
    <oa:CreationDateTime>2009-04-02T17:23:34.608Z</oa:CreationDateTime>
    <oa:BODID>fe70f2d0-1faa-11de-9dd0-827f49d4ef1b</oa:BODID>
  </oa:ApplicationArea>
  <_inv:DataArea>
    <oa:Acknowledge>
      <oa:OriginalApplicationArea>
        <oa:CreationDateTime>2009-04-02T08:28:47.718Z</oa:CreationDateTime>
        <oa:BODID>4929ef20-1f60-11de-8f9a-834749d4773b</oa:BODID>
      </oa:OriginalApplicationArea>
    </oa:Acknowledge>
    <_inv:InventoryRequirement shipAsComplete="false">
      <_ord:OrderIdentifier>
        <_wcf:UniqueID>14507</_wcf:UniqueID>
      </_ord:OrderIdentifier>
      <_ord:StoreIdentifier>
        <_wcf:UniqueID>553</_wcf:UniqueID>
      </_ord:StoreIdentifier>
      <_ord:BuyerIdentifier>
        <_wcf:UniqueID>100000000501</_wcf:UniqueID>
        <_wcf:DistinguishedName>uid=user,ou=extended sites seller organizationsample
b2c store,o=extended sites seller organization,o=root
organization</_wcf:DistinguishedName>
      </_ord:BuyerIdentifier>
      <_ord:OrderStatus prepareIndicator="false">
        <_ord:Status>P</_ord:Status>
      </_ord:OrderStatus>
      <_ord:LastUpdateDate>2009-01-13T09:57:33.420Z</_ord:LastUpdateDate>
      <_ord:PlacedDate>2009-01-13T09:57:33.420Z</_ord:PlacedDate>
      <_ord:OrderItem>
```

```

    <_ord:OrderItemIdentifier>
      <_wcf:UniqueID>10045007</_wcf:UniqueID>
    </_ord:OrderItemIdentifier>
    <_ord:OrderItemStatus>
      <_ord:Status>M</_ord:Status>
      <_ord:InventoryStatus>Unallocated</_ord:InventoryStatus>
    </_ord:OrderItemStatus>
    <_ord:OrderItemFulfillmentInfo/>
    <_ord:FulfillmentCenter>
      <_ord:FulfillmentCenterIdentifier>
        <_wcf:UniqueID/>
        <_wcf:Name/>
      </_ord:FulfillmentCenterIdentifier>
    </_ord:FulfillmentCenter>
  </_ord:OrderItem>
</_inv:InventoryRequirement>
</_inv:DataArea>
</_inv:AcknowledgeInventoryRequirement>

```

A.5 ProcessOrder with action code TransferOrder request

Example A-5 shows the ProcessOrder with the action code TransferOrder request.

Example A-5 ProcessOrder with action code TransferOrder request sample

```

<_ord:ProcessOrder releaseID="9.0" versionID="7.0.0.0"
xmlns:_ord="http://www.ibm.com/xmlns/prod/commerce/9/order"
xmlns:_wcf="http://www.ibm.com/xmlns/prod/commerce/9/foundation">
  <oa:ApplicationArea xsi:type="_wcf:ApplicationAreaType">
    <oa:CreationDateTime
xsi:type="udt:DateType">2009-08-11</oa:CreationDateTime>
    <oa:BODID>20aed710-8692-11de-845f-84214a818784</oa:BODID>
    <_wcf:BusinessContext/>
  </oa:ApplicationArea>
  <_ord:DataArea>
    <oa:Process>
      <oa:ActionCriteria>
        <oa:ActionExpression actionCode="TransferOrder"
expressionLanguage="_wcf:XPath">/Order</oa:ActionExpression>
      </oa:ActionCriteria>
    </oa:Process>
  </_ord:DataArea>
</_ord:ProcessOrder>

```

```

</oa:Process>
<_ord:Order shipAsComplete="true">
  <_ord:OrderIdentifier>
    <_wcf:UniqueID>12501</_wcf:UniqueID>
  </_ord:OrderIdentifier>
  <_ord:StoreIdentifier>
    <_wcf:UniqueID>10001</_wcf:UniqueID>
    <_wcf:ExternalIdentifier>
      <_wcf:NameIdentifier>Madisons</_wcf:NameIdentifier>
    </_wcf:ExternalIdentifier>
  </_ord:StoreIdentifier>
  <_ord:BuyerIdentifier>
    <_wcf:UniqueID>2</_wcf:UniqueID>
    <_wcf:DistinguishedName>uid=wangful,o=default
organization,o=root organization</_wcf:DistinguishedName>
  </_ord:BuyerIdentifier>
  <_ord:OrderAmount>
    <_wcf:GrandTotal currency="USD">323.99000</_wcf:GrandTotal>
    <_wcf:TotalProductPrice
currency="USD">449.99000</_wcf:TotalProductPrice>
    <_wcf:TotalAdjustment
currency="USD">-126.00000</_wcf:TotalAdjustment>
    <_wcf:Adjustment>
      <_wcf:Usage>Discount</_wcf:Usage>
      <_wcf:Code>Furniture Category Discount</_wcf:Code>
      <_wcf:Description language="-1">Save 20% on
Furniture!</_wcf:Description>
      <_wcf:Amount currency="USD">-90.00000</_wcf:Amount>
      <_wcf:DisplayLevel
xsi:type="_wcf:DisplayLevelEnumerationType">OrderItem</_wcf:DisplayLevel>
    </_wcf:Adjustment>
    <_wcf:Adjustment>
      <_wcf:Usage>Discount</_wcf:Usage>
      <_wcf:Code>Save 10% on all orders today</_wcf:Code>
      <_wcf:Description language="-1">Save 10% on all
orders</_wcf:Description>
      <_wcf:Amount currency="USD">-36.00000</_wcf:Amount>
      <_wcf:DisplayLevel
xsi:type="_wcf:DisplayLevelEnumerationType">Order</_wcf:DisplayLevel>
    </_wcf:Adjustment>
    <_wcf:TotalShippingCharge
currency="USD">0.00000</_wcf:TotalShippingCharge>
    <_wcf:TotalSalesTax
currency="USD">0.00000</_wcf:TotalSalesTax>

```

```

        <_wcf:TotalShippingTax
currency="USD">0.00000</_wcf:TotalShippingTax>
    </_ord:OrderAmount>
    <_ord:OrderPaymentInfo>
        <_ord:PaymentInstruction>
            <_ord:UniqueID>12001</_ord:UniqueID>
            <_ord:Amount currency="USD">323.99000</_ord:Amount>
            <_ord:PaymentMethod>

<_ord:PaymentMethodName>PayInStore</_ord:PaymentMethodName>
            <_ord:Description language="-1">Pay In
Store</_ord:Description>
            </_ord:PaymentMethod>
            <_ord:ProtocolData
name="payment_method">PayInStore</_ord:ProtocolData>
            </_ord:PaymentInstruction>
        </_ord:OrderPaymentInfo>
        <_ord:OrderStatus prepareIndicator="true">
            <_ord:Status
xsi:type="_ord:OrderLifeCycleStatusEnumerationType">M</_ord:Status>
        </_ord:OrderStatus>

<_ord:LastUpdateDate>2009-08-11T16:10:02.890Z</_ord:LastUpdateDate>
    <_ord:PlacedDate>2009-08-11T16:10:02.890Z</_ord:PlacedDate>
    <_ord:OrderItem>
        <_ord:OrderItemIdentifier>
            <_wcf:UniqueID>45001</_wcf:UniqueID>
        </_ord:OrderItemIdentifier>
        <_ord:CatalogEntryIdentifier>
            <_wcf:UniqueID>10002</_wcf:UniqueID>
            <_wcf:ExternalIdentifier>
                <_wcf:PartNumber>FUL0-0101</_wcf:PartNumber>
            </_wcf:ExternalIdentifier>
        </_ord:CatalogEntryIdentifier>
        <_ord:Quantity uom="C62">1.0</_ord:Quantity>
        <_ord:ContractIdentifier>
            <_wcf:UniqueID>10001</_wcf:UniqueID>
        </_ord:ContractIdentifier>
        <_ord:OfferID>10002</_ord:OfferID>
        <_ord:OrderItemAmount freeGift="false">
            <_wcf:UnitPrice>
                <_wcf:Price currency="USD">449.99000</_wcf:Price>
                <_wcf:Quantity uom="C62">1.0</_wcf:Quantity>
            </_wcf:UnitPrice>

```

```

        <_wcf:OrderItemPrice
currency="USD">449.99000</_wcf:OrderItemPrice>
        <_wcf:Adjustment>
            <_wcf:Usage>Discount</_wcf:Usage>
            <_wcf:Code>Furniture Category Discount</_wcf:Code>
            <_wcf:Description language="-1">Save 20% on
Furniture!</_wcf:Description>
            <_wcf:Amount currency="USD">-90.00000</_wcf:Amount>
            <_wcf:DisplayLevel
xsi:type="_wcf:DisplayLevelEnumerationType">OrderItem</_wcf:DisplayLevel>
        </_wcf:Adjustment>
        <_wcf:Adjustment>
            <_wcf:Usage>Discount</_wcf:Usage>
            <_wcf:Code>Save 10% on all orders today</_wcf:Code>
            <_wcf:Description language="-1">Save 10% on all
orders</_wcf:Description>
            <_wcf:Amount currency="USD">-36.00000</_wcf:Amount>
            <_wcf:DisplayLevel
xsi:type="_wcf:DisplayLevelEnumerationType">Order</_wcf:DisplayLevel>
        </_wcf:Adjustment>
        <_wcf:ShippingCharge
currency="USD">0.00000</_wcf:ShippingCharge>
        <_wcf:SalesTax currency="USD">0.00000</_wcf:SalesTax>
        <_wcf:ShippingTax currency="USD">0.00000</_wcf:ShippingTax>
    </_ord:OrderItemAmount>
    <_ord:OrderItemShippingInfo expedite="false">
        <_ord:ShippingAddress>
            <_wcf:ContactInfoIdentifier>
                <_wcf:UniqueID>10076</_wcf:UniqueID>
                <_wcf:ExternalIdentifier>
                    <_wcf:ContactInfoNickName>Calgary
Mall</_wcf:ContactInfoNickName>
                </_wcf:ExternalIdentifier>
            </_wcf:ContactInfoIdentifier>
            <_wcf:ContactName>
                <_wcf:PersonTitle></_wcf:PersonTitle>
                <_wcf:LastName>Calgary Mall</_wcf:LastName>
                <_wcf:FirstName></_wcf:FirstName>
                <_wcf:MiddleName></_wcf:MiddleName>
            </_wcf:ContactName>
            <_wcf:Address>
                <_wcf:AddressLine>1025 Cameron Ave
SW</_wcf:AddressLine>
                <_wcf:AddressLine></_wcf:AddressLine>
            </_wcf:Address>
        </_ord:ShippingAddress>
    </_ord:OrderItemShippingInfo>
</_ord:OrderItem>
</_ord:Order>
</_wcf:Order>

```

```

        <_wcf:AddressLine></_wcf:AddressLine>
        <_wcf:City> Calgary</_wcf:City>

    <_wcf:StateOrProvinceName>Alberta</_wcf:StateOrProvinceName>
        <_wcf:Country>Canada</_wcf:Country>
        <_wcf:PostalCode>T2T 0K4</_wcf:PostalCode>
    </_wcf:Address>
    <_wcf:Telephone1 publish="true">
        <_wcf:Value></_wcf:Value>
    </_wcf:Telephone1>
    <_wcf:Telephone2 publish="true">
        <_wcf:Value></_wcf:Value>
    </_wcf:Telephone2>
    <_wcf:EmailAddress1>
        <_wcf:Value>admin@madisons.ca</_wcf:Value>
    </_wcf:EmailAddress1>
    <_wcf:EmailAddress2>
        <_wcf:Value></_wcf:Value>
    </_wcf:EmailAddress2>
    <_wcf:Fax1>
        <_wcf:Value></_wcf:Value>
    </_wcf:Fax1>
    <_wcf:Fax2>
        <_wcf:Value></_wcf:Value>
    </_wcf:Fax2>
</_ord:ShippingAddress>
<_ord:ShippingMode>
    <_ord:ShippingModeIdentifier>
        <_ord:UniqueID>10001</_ord:UniqueID>
        <_ord:ExternalIdentifier>
            <_ord:ShipModeCode>PickupInStore</_ord:ShipModeCode>
        </_ord:ExternalIdentifier>
    </_ord:ShippingModeIdentifier>
    <_ord:Description language="-1">Pickup in
store</_ord:Description>
</_ord:ShippingMode>
<_ord:PhysicalStoreIdentifier>
    <_wcf:UniqueID>10026</_wcf:UniqueID>
    <_wcf:ExternalIdentifier>Calgary
Mall</_wcf:ExternalIdentifier>
</_ord:PhysicalStoreIdentifier>
</_ord:OrderItemShippingInfo>
<_ord:OrderItemStatus>
    <_ord:Status
xsi:type="_ord:OrderItemLifeCycleStatusEnumerationType">M</_ord:Status>

```

```

        <_ord:InventoryStatus
xsi:type="_ord:InventoryStatusEnumerationType">Allocated</_ord:InventoryStatus>
        <_ord:FulfillmentStatus
xsi:type="_ord:FulfillmentStatusEnumerationType">Unreleased</_ord:FulfillmentStatus>
        </_ord:OrderItemStatus>
        <_ord:OrderItemFulfillmentInfo>

<_ord:AvailableDate>2009-08-11T16:09:31.468Z</_ord:AvailableDate>

<_ord:ExpectedShipDate>2009-08-11T16:15:03.640Z</_ord:ExpectedShipDate>
        </_ord:OrderItemFulfillmentInfo>
        <_ord:FulfillmentCenter>
        <_ord:FulfillmentCenterIdentifier>
        <_wcf:UniqueID>10076</_wcf:UniqueID>
        <_wcf:Name>Calgary Mall</_wcf:Name>
        </_ord:FulfillmentCenterIdentifier>
        </_ord:FulfillmentCenter>
        <_ord:CorrelationGroup>45001</_ord:CorrelationGroup>
        <_ord:CreateDate>2009-08-11T15:36:06.781Z</_ord:CreateDate>

<_ord:LastUpdateDate>2009-08-11T16:10:02.890Z</_ord:LastUpdateDate>
        <_ord:UsableShippingChargePolicy>
        <_wcf:UniqueID>-7001</_wcf:UniqueID>
        <_wcf:ExternalIdentifier>
        <_wcf:Name>StandardShippingChargeBySeller</_wcf:Name>
        <_wcf:Type>ShippingCharge</_wcf:Type>
        </_wcf:ExternalIdentifier>
        </_ord:UsableShippingChargePolicy>
        </_ord:OrderItem>
    </_ord:Order>
</_ord:DataArea>
</_ord:ProcessOrder>

```

A.6 ProcessOrder with action code TransferOrder response

Example A-6 shows the ProcessOrder with the action code TransferOrder response.

Example A-6 ProcessOrder with action code TransferOrder response

```
<_ord:AcknowledgeOrder
xmlns:_ord="http://www.ibm.com/xmlns/prod/commerce/9/order"
xmlns:_wcf="http://www.ibm.com/xmlns/prod/commerce/9/foundation">
  <oa:ApplicationArea>
    <oa:CreationDateTime
xsi:type="udt:DateTimeType">2009-08-11T16:15:06Z</oa:CreationDateTime>
  </oa:ApplicationArea>
  <_ord:DataArea>
    <oa:Acknowledge/>
    <_ord:Order>
      <_ord:OrderIdentifier>
        <_wcf:UniqueID>12501</_wcf:UniqueID>
      </_ord:OrderIdentifier>
    </_ord:Order>
  </_ord:DataArea>
</_ord:AcknowledgeOrder>
```

A.7 SyncOrder request

Example A-7 shows a SyncOrder request.

Example A-7 SyncOrder request

```
<_ord:SyncOrder releaseID="9.0" versionID="6.0.0.6"
xmlns:_ord="http://www.ibm.com/xmlns/prod/commerce/9/order"
xmlns:_wcf="http://www.ibm.com/xmlns/prod/commerce/9/foundation"
xmlns:oa="http://www.openapplications.org/oagis/9">
  <oa:ApplicationArea xsi:type="_wcf:ApplicationAreaType">
    <oa:CreationDateTime>2009-08-06T02:36:47.578Z</oa:CreationDateTime>
    <oa:BODID>fc925370-8231-11de-b571-81ff4a7a41bf</oa:BODID>
    <_wcf:BusinessContext>
      <_wcf:ContextData name="langId">-1</_wcf:ContextData>
      <_wcf:ContextData name="storeId">0</_wcf:ContextData>
    </_wcf:BusinessContext>
  </oa:ApplicationArea>
```



```

    <_ord:DataArea>
      <oa:Sync>
        <oa:ActionCriteria>
          <oa:ActionExpression actionCode="Change"
expressionLanguage="_wcf:XPath"/>
        </oa:ActionCriteria>
      </oa:Sync>
    <_ord:Order>
      <_ord:OrderIdentifier>
        <_wcf:UniqueID>17010</_wcf:UniqueID>
      </_ord:OrderIdentifier>
      <_ord:OrderAmount/>
      <_ord:OrderStatus>
        <_ord:Status>E</_ord:Status>
      </_ord:OrderStatus>
      <_ord:OrderItem>
        <_ord:OrderItemIdentifier>
          <_wcf:UniqueID>10070050</_wcf:UniqueID>
        </_ord:OrderItemIdentifier>
        <_ord:OrderItemAmount/>
        <_ord:OrderItemShippingInfo/>
        <_ord:OrderItemStatus>
          <_ord:Status>E</_ord:Status>
        </_ord:OrderItemStatus>
      </_ord:OrderItem>
      <_wcf:UserData/>
    </_ord:Order>
  </_ord:DataArea>
</_ord:SyncOrder>

```

A.8 SyncOrder response

Example A-8 shows the contents of the ProcessInventoryReqmt_UpdateInventoryReservationReq.xml file.

Example A-8 SyncOrder response

```

<oa:ConfirmBOD releaseID="9.0"
xmlns:_wcf="http://www.ibm.com/xmlns/prod/commerce/9/foundation"
xmlns:oa="http://www.openapplications.org/oagis/9">
  <oa:ApplicationArea xsi:type="_wcf:ApplicationAreaType">
    <oa:CreationDateTime>2009-08-06T16:44:37.862Z</oa:CreationDateTime>
    <oa:BODID>6d720880-82a8-11de-85cd-861d4a788c3a</oa:BODID>
  </oa:ApplicationArea>
</oa:ConfirmBOD>

```

```

</oa:ApplicationArea>
<oa:DataArea>
  <oa:Confirm>
    <oa:OriginalApplicationArea>
      <oa:CreationDateTime>2009-08-06Z</oa:CreationDateTime>
      <oa:BODID>fc925370-8231-11de-b571-81ff4a7a41bf</oa:BODID>
    </oa:OriginalApplicationArea>
  </oa:Confirm>
  <oa:BOD>
    <oa:OriginalApplicationArea xsi:type="_wcf:ApplicationAreaType">
      <oa:CreationDateTime>2009-08-06Z</oa:CreationDateTime>
      <oa:BODID>fc925370-8231-11de-b571-81ff4a7a41bf</oa:BODID>
      <_wcf:BusinessContext>
        <_wcf:ContextData name="langId">-1</_wcf:ContextData>
        <_wcf:ContextData name="storeId">0</_wcf:ContextData>
      </_wcf:BusinessContext>
    </oa:OriginalApplicationArea>
    <oa:BODSuccessMessage/>
  </oa:BOD>
</oa:DataArea>
</oa:ConfirmBOD>

```

A.9 ProcessInventoryReqmt_UpdateInventoryReservationsReq.xsl

Example A-9 shows the content of the
ProcessInventoryReqmt_UpdateInventoryReservationsReq.xsl file.

Example A-9 ProcessInventoryReqmt_UpdateInventoryReservationsReq.xsl

```
<?xml version="1.0" encoding="UTF-8"?>

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xalan="http://xml.apache.org/xalan"
xmlns:datetime="http://exslt.org/dates-and-times"
xmlns:oa="http://www.openapplications.org/oagis/9"
xmlns:udt="http://www.openapplications.org/oagis/9/unqualifieddatatypes
/1.1" xmlns:_wcf="http://www.ibm.com/xmlns/prod/commerce/9/foundation"
xmlns:_sto="http://www.ibm.com/xmlns/prod/commerce/9/store"
xmlns:_inv="http://www.ibm.com/xmlns/prod/commerce/9/inventory"
xmlns:_ord="http://www.ibm.com/xmlns/prod/commerce/9/order"
xmlns:test="http://test" exclude-result-prefixes="xalan oa udt _wcf
_sto _inv _ord" version="1.0">
  <xsl:output method="xml" encoding="UTF-8" indent="yes"
xalan:indent-amount="2" />
  <xsl:strip-space elements="*" />
  <xsl:template match="/">
    <xsl:variable name="order"
select="_inv:ProcessInventoryRequirement/_inv:DataArea/_inv:InventoryRe
quirement" />
    <test:updateInventoryReservations>
      <order>
        <orderId><xsl:value-of
select="$order/_ord:OrderIdentifier/_wcf:UniqueID" /></orderId>
        <description xsi:nil="true" />
        <storeId><xsl:value-of
select="$order/_ord:StoreIdentifier/_wcf:UniqueID" /></storeId>
        <userDn><xsl:value-of
select="$order/_ord:BuyerIdentifier/_wcf:DistinguishedName" /></userDn>
        <status><xsl:value-of
select="$order/_ord:OrderStatus/_ord:Status" /></status>
        <xsl:choose>
          <xsl:when test="$order/_ord:PlacedDate">
```

```

        <submissionDate><xsl:value-of
select="$order/_ord:PlacedDate" /></submissionDate>
        </xsl:when>
        <xsl:otherwise>
        <submissionDate xsi:nil="true" />
        </xsl:otherwise>
    </xsl:choose>
    <orderItems>
        <xsl:for-each select="$order/_ord:OrderItem">
            <OrderItem>
                <orderItemId><xsl:value-of
select="_ord:OrderItemIdentifier/_wcf:UniqueID" /></orderItemId>
                <sku><xsl:value-of
select="_ord:CatalogEntryIdentifier/_wcf:ExternalIdentifier/_wcf:PartNu
mber" /></sku>
                <quantity>
                    <value><xsl:value-of select="_ord:Quantity"
/></value>
                    <xsl:choose>
                        <xsl:when test="_ord:Quantity/@uom">
                            <uom><xsl:value-of
select="_ord:Quantity/@uom" /></uom>
                        </xsl:when>
                        <xsl:otherwise>
                            <uom>C62</uom>
                        </xsl:otherwise>
                    </xsl:choose>
                </quantity>
                <shippingModeId><xsl:value-of
select="_ord:OrderItemShippingInfo/_ord:ShippingMode/_ord:ShippingModeI
dentifier/_ord:ExternalIdentifier/_ord:ShipModeCode"
/></shippingModeId>
                <fulfillmentCenterId><xsl:value-of
select="_ord:FulfillmentCenter/_ord:FulfillmentCenterIdentifier/_wcf:Na
me" /></fulfillmentCenterId>
                <xsl:choose>
                    <xsl:when
test="_ord:OrderItemShippingInfo/_ord:RequestedShipDate">
                        <requestedShipDate><xsl:value-of
select="_ord:OrderItemShippingInfo/_ord:RequestedShipDate"
/></requestedShipDate>
                    </xsl:when>
                    <xsl:otherwise>
                        <requestedShipDate xsi:nil="true" />
                    </xsl:otherwise>
                </xsl:choose>
            </OrderItem>
        </xsl:for-each>
    </orderItems>

```

```

        </xsl:choose>
        <status><xsl:value-of
select="_ord:OrderItemStatus/_ord:Status" /></status>
        <inventoryStatus xsi:nil="true" />
        <availableDate xsi:nil="true" />
        <shipDate xsi:nil="true" />
        <attribute>
        <!-- temporary -->
        <Attribute>
        <name>wcFulfillmentCenterId</name>
        <value><xsl:value-of
select="_ord:FulfillmentCenter/_ord:FulfillmentCenterIdentifier/_wcf:Un
iqueID" /></value>
        </Attribute>
        </attribute>
        </OrderItem>
    </xsl:for-each>
</orderItems>
</order>
</test:updateInventoryReservations>
</xsl:template>
</xsl:stylesheet>

```

A.10 UpdateInventoryReservationsResp_AcknowledgeInventoryReqmt.xsl

Example A-10 shows the contents of the UpdateInventoryReservationsResp_AcknowledgeInventoryReqmt.xsl file.

Example A-10 UpdateInventoryReservationsResp_AcknowledgeInventoryReqmt.xsl

```
<?xml version="1.0" encoding="UTF-8"?>

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xalan="http://xml.apache.org/xalan"
xmlns:datetime="http://exslt.org/dates-and-times"
xmlns:oa="http://www.openapplications.org/oagis/9"
xmlns:udt="http://www.openapplications.org/oagis/9/unqualifieddatatypes
/1.1" xmlns:_wcf="http://www.ibm.com/xmlns/prod/commerce/9/foundation"
xmlns:_sto="http://www.ibm.com/xmlns/prod/commerce/9/store"
xmlns:_inv="http://www.ibm.com/xmlns/prod/commerce/9/inventory"
xmlns:_ord="http://www.ibm.com/xmlns/prod/commerce/9/order"
xmlns:test="http://test" exclude-result-prefixes="xalan test"
version="1.0">
  <xsl:output method="xml" encoding="UTF-8" indent="yes"
xalan:indent-amount="2" />
  <xsl:strip-space elements="*" />
  <xsl:template match="/">
    <xsl:variable name="order"
select="test:updateInventoryReservationsResponse/updateInventoryReserva
tionsReturn" />
    <_inv:AcknowledgeInventoryRequirement>
      <oa:ApplicationArea>
        <oa:CreationDateTime
xsi:type="udt:DateTimeType"><xsl:value-of select="datetime:dateTime()"
/></oa:CreationDateTime>
      </oa:ApplicationArea>
      <_inv:DataArea>
        <oa:Acknowledge />
        <_inv:InventoryRequirement>
          <_ord:OrderIdentifier>
            <_wcf:UniqueID><xsl:value-of select="$order/orderId"
/></_wcf:UniqueID>
          </_ord:OrderIdentifier>
          <xsl:for-each select="$order/orderItems/OrderItem">
```

```

        <_ord:OrderItem>
          <_ord:OrderItemIdentifier>
            <_wcf:UniqueID><xsl:value-of
select="orderId" /></_wcf:UniqueID>
          </_ord:OrderItemIdentifier>
          <_ord:OrderItemStatus>
            <_ord:Status><xsl:value-of select="status"
/></_ord:Status>
            <xsl:choose>
              <xsl:when test="inventoryStatus='A'">
                <_ord:InventoryStatus>Allocated</_ord:InventoryStatus>
              </xsl:when>
              <xsl:when test="inventoryStatus='B'">
                <_ord:InventoryStatus>Backordered</_ord:InventoryStatus>
              </xsl:when>
              <xsl:otherwise>
                <_ord:InventoryStatus>Unallocated</_ord:InventoryStatus>
              </xsl:otherwise>
            </xsl:choose>
          </_ord:OrderItemStatus>
          <_ord:OrderItemFulfillmentInfo>
            <xsl:if
test="availableDate[not(@xsi:nil='true')]">
              <_ord:AvailableDate><xsl:value-of
select="availableDate" /></_ord:AvailableDate>
            </xsl:if>
            <xsl:if test="shipDate[not(@xsi:nil='true')]">
              <_ord:ExpectedShipDate><xsl:value-of
select="shipDate" /></_ord:ExpectedShipDate>
            </xsl:if>
          </_ord:OrderItemFulfillmentInfo>
          <xsl:if
test="fulfillmentCenterId[not(@xsi:nil='true')]">
            <_ord:FulfillmentCenter>
              <_ord:FulfillmentCenterIdentifier>
                <!-- temporary -->
                <_wcf:UniqueID><xsl:value-of
select="attribute/Attribute[name='wcFulfillmentCenterId']/value"
/></_wcf:UniqueID>
                <_wcf:Name><xsl:value-of
select="fulfillmentCenterId" /></_wcf:Name>
              </_ord:FulfillmentCenterIdentifier>
            </_ord:FulfillmentCenter>
          </xsl:if>
        </_ord:OrderItem>

```

```
        </xsl:if>
      </_ord:OrderItem>
    </xsl:for-each>
  </_inv:InventoryRequirement>
</_inv:DataArea>
</_inv:AcknowledgeInventoryRequirement>
</xsl:template>
</xsl:stylesheet>
```



B

Sample code

This appendix includes various sample codes that you can use with the examples Chapter 5, “Mobile commerce features in WebSphere Commerce V7” on page 177.

B.1 ReviewsDataBean sample code

Example B-1 shows the content of the ReviewsDataBean sample code.

Example B-1 ReviewsDataBean Sample code

```
package com.ibm.itso.commerce.socom.beans;

import java.util.LinkedHashMap;
import java.util.LinkedList;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;

import org.apache.commons.httpclient.HttpClient;
import org.apache.commons.httpclient.NameValuePair;
import org.apache.commons.httpclient.methods.GetMethod;
import org.apache.commons.httpclient.util.EncodingUtil;
import org.json.JSONArray;
import org.json.JSONObject;

import com.ibm.commerce.beans.SmartDataBean;
import com.ibm.commerce.beans.SmartDataBeanImpl;

public class ReviewsDataBean extends SmartDataBeanImpl implements
SmartDataBean {

    private static final int INT_NO_SET = -1;

    //pageSize sets the number of reviews to be displayed on a page
    private int pageSize = INT_NO_SET;
    //onPage sets the page number
    private int onPage = INT_NO_SET;
    //sortOptions sets the sort type & sort direction of the results.
    //Its format is sort type|sort direction. Valid values for sort type
are "rating" and "submissionTime". Valid values for sort direction are
"asc" & "desc".
    private String sortOptions;
    //partNumber sets the Part number for which results are needed.
    private String partNumber;

    //numOfOverallRatings - total number of ratings for the product.
    private int numOfOverallRatings;
    //this is average overall rating for the product.
    private double avgOverallRating;
```

```

//Lower bound used to calculate the average rating.
private int lowerBound;
//Upper bound used to calculate the average rating
private int upperBound;
//Sorted map of reviewId and ReviewDetailBean
private LinkedHashMap<String, ReviewDetailBean> reviewDetailsMap;
//Sort direction used for sorting
private String sortDirection;
//Number of results returned by sMash application, to be displayed
on a page. - used in Pagination
private int noOfEntriesPerPage;
//Total number of reviews for the product - used in Pagination
private int noOfTotalEntries;
//Current page number - used in pagination
private int pageNum;
//Sort type used for sorting
private String sortType;
//Number of total pages - Used in pagination
private int noOfTotalPages;

private String errorMessage = null;
private Throwable errorException = null;

private static final String CLASS_NAME =
ReviewsDataBean.class.getName();
private static final Logger LOGGER = Logger.getLogger(CLASS_NAME);

public ReviewsDataBean() {
    reviewDetailsMap = new LinkedHashMap<String, ReviewDetailBean>();
}

private void setErrorState(String error, Throwable t) {
    errorMessage = error;
    errorException = t;
}

public boolean isError() {
    return errorMessage != null;
}

public String getErrorMessage() {
    return errorMessage;
}

public Throwable getErrorException() {

```

```

        return errorException;
    }

    public int getNumOfOverallRatings() {
        return numOfOverallRatings;
    }

    public double getAvgOverallRating() {
        return avgOverallRating;
    }

    public int getLowerBound() {
        return lowerBound;
    }

    public int getUpperBound() {
        return upperBound;
    }

    public void setPageSize(int pageSize) {
        this.pageSize = pageSize;
    }

    public void setOnPage(int onPage) {
        this.onPage = onPage;
    }

    public void setSortOptions(String sortOptions) {
        this.sortOptions = sortOptions;
    }

    public void setPartNumber(String partNumber) {
        this.partNumber = partNumber;
    }

    public LinkedHashMap<String, ReviewDetailBean> getReviewDetailsMap()
    {
        return reviewDetailsMap;
    }

    public String getSortDirection() {
        return sortDirection;
    }

    public int getNoOfEntriesPerPage() {

```

```

        return noOfEntriesPerPage;
    }

    public int getNoOfTotalEntries() {
        return noOfTotalEntries;
    }

    public int getPageNum() {
        return pageNum;
    }

    public String getSortType() {
        return sortType;
    }

    public int getNoOfTotalPages() {
        return noOfTotalPages;
    }

    public void populate() {
        final String METHOD_NAME = "populate";
        LOGGER.entering(CLASS_NAME, METHOD_NAME);
        errorMessage = null;
        errorException = null;
        String rating = "";
        try {
            if (LOGGER.isLoggable(Level.FINE)) {
                LOGGER.logp(Level.FINE, CLASS_NAME, METHOD_NAME, "partNumber:
\\\"{0}\\\", pageSize: \\\"{1}\\\", onPage: \\\"{2}\\\", sortOptions: \\\"{3}\\\", new
String[] { partNumber, String.valueOf(pageSize),
String.valueOf(onPage), sortOptions });
            }
            if (partNumber != null) {

                //Read protocol and host name from properties
                String urlString = http://localhost/soccom/api/items/"+
partNumber + "/reviews";
                if (LOGGER.isLoggable(Level.FINE)) {
                    LOGGER.logp(Level.FINE, CLASS_NAME, METHOD_NAME, "URL:
\\\"{0}\\\", urlString);
                }
                HttpClient client = new HttpClient();
                client.getParams().setParameter("http.socket.timeout",
2000);
            }
        } catch (Exception e) {
            errorMessage = e.getMessage();
            errorException = e;
        }
        LOGGER.exiting(CLASS_NAME, METHOD_NAME);
    }
}

```

```

client.getParams().setParameter("http.protocol.content-charset",
"utf-8");

// Create a method instance.
GetMethod method = new GetMethod(urlString);

List<NameValuePair> params = new
LinkedList<NameValuePair>();
    if (pageSize != INT_NO_SET) {
        params.add(new NameValuePair("pageSize",
String.valueOf(pageSize)));
    }
    if (onPage != INT_NO_SET) {
        params.add(new NameValuePair("onPage",
String.valueOf(onPage)));
    }
    params.add(new NameValuePair("sortOptions", sortOptions));

NameValuePair[] nvpArr = params.toArray(new
NameValuePair[params.size()]);

method.setQueryString(EncodingUtil.formUrlEncode(nvpArr,
"utf-8"));

method.getParams().setParameter("http.socket.timeout",
2000);

// Execute
int statuscode = client.executeMethod(method);

if (LOGGER.isLoggable(Level.FINE)) {
    LOGGER.logp(Level.FINE, CLASS_NAME, METHOD_NAME,
        "HTTP status code: {0}", statuscode);
}

if (statuscode >= 200 && statuscode < 400) {
    // Read the response body.
    byte[] responseBody = method.getResponseBody();

    rating = new String(responseBody, "UTF-8");

    if (LOGGER.isLoggable(Level.FINE)) {
        LOGGER.logp(Level.FINE, CLASS_NAME, METHOD_NAME,
            "HTTP response: \"{0}\"", rating);
    }
}

```

```

        }

    }

    method.releaseConnection();
}

} catch (Exception e) {
    LOGGER.logp(Level.SEVERE, CLASS_NAME, METHOD_NAME,
        "Exception in trying to retrieve review details", e);
    setErrorState("Exception in trying to retrieve review
details", e);
}
try {
    if (rating != null && !rating.equals("")) {
        JSONObject json = new JSONObject(rating);
        if (json.getInt("status") == 200) {
            JSONObject data = json.getJSONObject("data");

            JSONObject ratingStatistics =
data.getJSONObject("ratingStatistics");
            numOfOverallRatings =
ratingStatistics.getInt("numOfOverallRatings");
            if (numOfOverallRatings > 0) {
                avgOverallRating =
ratingStatistics.getDouble("avgOverallRating");
            }
            JSONObject ratingRange =
ratingStatistics.getJSONObject("ratingRange");
            upperBound = ratingRange.getInt("upperBound");
            lowerBound = ratingRange.getInt("lowerBound");

            JSONObject paginationDetails =
data.getJSONObject("paginationDetails");
            sortDirection =
paginationDetails.getString("sortDirection");
            noOfEntriesPerPage =
paginationDetails.getInt("noOfEntriesPerPage");
            noOfTotalEntries =
paginationDetails.getInt("noOfTotalEntries");
            pageNum = paginationDetails.getInt("pageNum");
            sortType = paginationDetails.getString("sortType");
            noOfTotalPages =
paginationDetails.getInt("noOfTotalPages");

            JSONArray reviews = data.getJSONArray("reviews");

```

```

        if (reviews.length() > 0) {
            JSONObject review = null;
            ReviewDetailBean reviewDetails = null;
            for (int i = 0; i < reviews.length(); i++) {
                review = reviews.getJSONObject(i);
                reviewDetails = new ReviewDetailBean();

reviewDetails.setRatingOnly(review.getBoolean("isRatingOnly"));

reviewDetails.setItemId(review.getString("itemId"));
                reviewDetails.setRating(review.getInt("rating"));
                int reviewId = review.getInt("reviewId");
                reviewDetails.setReviewId(reviewId);

reviewDetails.setReviewText(review.getString("reviewText"));

reviewDetails.setSubmissionTime(review.getString("submissionTime"));
                reviewDetails.setTitle(review.getString("title"));

reviewDetails.setUserId(review.getString("userId"));
                reviewDetailsMap.put(String.valueOf(reviewId),
reviewDetails);
            }
        }
    } catch (Exception e) {
        LOGGER.logp(Level.SEVERE, CLASS_NAME, METHOD_NAME, "Exception
in trying to parse review details", e);
        setErrorState("Exception in trying to parse review details",
e);
    }
}
}

```

B.2 ReviewsDetailBean sample code

Example B-2 shows the content of the ReviewsDetailBean sample code.

Example B-2 ReviewDetailBean Sample code

```
package com.ibm.itso.commerce.socom.beans;

import java.text.ParseException;
import java.util.Date;

public class ReviewDetailBean {
    private boolean isRatingOnly;
    private int reviewId;
    private String reviewText;
    private String userId;
    private String title;
    private String submissionTime;
    private int rating;
    private String itemId;
    private String toStringCache = null;

    public boolean isRatingOnly() {
        return isRatingOnly;
    }
    public void setRatingOnly(boolean isRatingOnly) {
        this.isRatingOnly = isRatingOnly;
        toStringCache = null;
    }
    public int getReviewId() {
        return reviewId;
    }
    public void setReviewId(int reviewId) {
        this.reviewId = reviewId;
        toStringCache = null;
    }
    public String getReviewText() {
        return reviewText;
    }
    public void setReviewText(String reviewText) {
        this.reviewText = reviewText;
        toStringCache = null;
    }
    public String getUserId() {
        return userId;
    }
}
```

```

    }
    public void setUserId(String userId) {
        this.userId = userId;
        toStringCache = null;
    }
    public String getTitle() {
        return title;
    }
    public void setTitle(String title) {
        this.title = title;
        toStringCache = null;
    }
    public String getSubmissionTime() {
        return submissionTime;
    }
    public Date getSubmissionTimeAsDate()
    {
        java.text.SimpleDateFormat sdf = new
java.text.SimpleDateFormat("yyyy-MM-dd'T'hh:mm:ss.SSS");

        try {
            java.util.Date d = sdf.parse(getSubmissionTime());
            return d;
        } catch (ParseException e) {
            return null;
        }
    }
    public void setSubmissionTime(String submissionTime) {
        this.submissionTime = submissionTime;
        toStringCache = null;
    }
    public int getRating() {
        return rating;
    }
    public void setRating(int rating) {
        this.rating = rating;
        toStringCache = null;
    }
    public String getItemId() {
        return itemId;
    }
    public void setItemId(String itemId) {
        this.itemId = itemId;
        toStringCache = null;
    }
}

```

```

public String toString() {
    if (toStringCache == null) {
        StringBuffer buf = new StringBuffer();

        buf.append("{");
        buf.append("reviewId: ");
        buf.append(reviewId);
        buf.append(", itemId: ");
        buf.append(itemId);
        buf.append(", userId: ");
        buf.append(userId);
        buf.append(", submissionTime: ");
        buf.append(submissionTime);
        buf.append(", rating: ");
        buf.append(rating);
        buf.append(", title: ");
        buf.append(title);
        buf.append(", reviewText: ");
        buf.append(reviewText);
        buf.append("}");

        toStringCache = buf.toString();
    }

    return toStringCache;
}

```

B.3 ReviewsDataBean sample code

Example B-3 shows the content of the ProductReviewList.jsp sample code.

Example B-3 Sample code for ProductReviewList.jsp

```
<%--
    *****
    * This JSP displays the product review list. It lets user sort the
    reviews based on rating or submission time.
    * It also has Pagination capabilities built in it.
    *****
--%>

<!-- BEGIN ProductReviewList.jsp -->

<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="http://commerce.ibm.com/base" prefix="wcbase" %>
<%@ taglib uri="http://commerce.ibm.com/foundation" prefix="wcf" %>

<%@ include file="../../../../include/parameters.jspf" %>
<%@ include file="../../../../include/JSTLEnvironmentSetup.jspf" %>

<c:set var="productId" value="${WCPParam.productId}" />

<c:set var="pgGrp" value="${WCPParam.pgGrp}" />

<wcbase:useBean id="catalogEntry"
classname="com.ibm.commerce.catalog.beans.CatalogEntryDataBean" />

<c:choose>
    <c:when test="${catalogEntry.bundle == true}">
        <wcbase:useBean id="product"
classname="com.ibm.commerce.catalog.beans.BundleDataBean"
scope="request" />
    </c:when>
    <c:when test="${catalogEntry.package == true}">
        <wcbase:useBean id="product"
classname="com.ibm.commerce.catalog.beans.PackageDataBean"
scope="request" />
    </c:when>
</c:choose>
```

```

        <c:when test="${catalogEntry.item == true}">
            <wcbase:useBean id="product"
classname="com.ibm.commerce.catalog.beans.ItemDataBean" scope="request"
/>
        </c:when>
        <c:when test="${catalogEntry.product == true}">
            <wcbase:useBean id="product"
classname="com.ibm.commerce.catalog.beans.ProductDataBean"
scope="request" />
        </c:when>
    </c:choose>
    <c:set property="partNumber" value="${catalogEntry.partNumber}"
target="${WCParam}" />
    <c:set var="productPage" value="true" scope="request" />

    <wcbase:useBean id="reviews"
classname="com.ibm.itso.commerce.socom.beans.ReviewsDataBean">
        <c:set property="pageSize" value="5" target="${reviews}" />
        <c:set property="onPage" value="${WCParam.onPage}"
target="${reviews}" />
        <c:set property="sortOptions" value="${WCParam.sortOptions}"
target="${reviews}" />
        <c:set property="partNumber" value="${WCParam.partNumber}"
target="${reviews}" />
    </wcbase:useBean>
    <!DOCTYPE html PUBLIC "-//WAPFORUM//DTD XHTML Mobile 1.2//EN"
"http://www.openmobilealliance.org/tech/DTD/xhtml-mobile12.dtd">

    <html xmlns="http://www.w3.org/1999/xhtml" lang="${shortLocale}"
xml:lang="${shortLocale}">
        <head>
            <title><fmt:message key="PRODUCT_TITLE" bundle="${storeText}" />:
<c:out value="${catalogEntry.description.name}"
escapeXml="false" /></title>
            <meta http-equiv="content-type" content="application/xhtml+xml"
/>
            <meta http-equiv="cache-control" content="max-age=300" />
            <meta name="viewport" content="width=device-width,
initial-scale=1.0, user-scalable=no" />
            <link rel="stylesheet" href="${cssPath}" />
        </head>

        <body>
            <div id="wrapper">

```

```

        <%@ include file="../../../include/HeaderDisplay.jspf" %>
        <%@ include
file="../../../include/BreadCrumbTrailDisplay.jspf"%>

        <%out.flush();%>

        <div id="reviews" class="content_box">
            <div class="heading_container">
                <h2>Reviews</h2>
                <div class="clear_float"></div>
            </div>
            <c:choose>
                <c:when test="${reviews.error}">
                    <li><fmt:message key="RATING_OVERALL_ERROR"
bundle="${storeText}" /></li>
                </c:when>
                <c:otherwise>
                    <a href="javascript:void(0);"><c:out
value="${product.description.name}" escapeXml="false"/></a>
                    <p>Overall Rating:
                    <c:import
url="${jspStoreDir}mobile/Snippets/ReusableObjects/StarRating.jsp">
                        <c:param name="lowerBound"
value="${reviews.lowerBound}" />
                        <c:param name="upperBound"
value="${reviews.upperBound}" />
                        <c:param name="avgOverallRating"
value="${reviews.avgOverallRating}" />
                    </c:import>
                    </p>
                    <c:choose>
                        <c:when test="${userType eq 'G'}">
                            <wcf:url var="logOnURL" value="MobileLogonForm">
                                <wcf:param name="catalogId"
value="${WParam.catalogId}" />
                                <wcf:param name="storeId"
value="${WParam.storeId}" />
                                <wcf:param name="langId" value="${WParam.langId}"
/>
                            </wcf:url>
                            <wcf:url var="registerURL"
value="MobileUserRegistrationAddForm">
                                <wcf:param name="catalogId"
value="${WParam.catalogId}" />

```

```

        <wcf:param name="storeId"
value="\${WCPParam.storeId}" />
        <wcf:param name="langId" value="\${WCPParam.langId}"
/>
        <wcf:param name="register_button" value="Register"
/>
    </wcf:url>
    <a href="\${fn:escapeXml(registerURL)}">Register</a>
to create a new review or <a href="\${fn:escapeXml(logOnURL)}"
title="<fmt:message key="SIGN_IN" bundle="\${storeText}"
/>"><fmt:message key="SIGN_IN" bundle="\${storeText}" /></a> in if you
are already a member.
    </c:when>
    <c:otherwise>

        <form method="get" action="mPostReviewsView">
            <fieldset>
                <c:forEach var="parameter"
items="\${WCPParamValues}">
                    <c:forEach var="value"
items="\${parameter.value}">
                        <input type="hidden" name="\${parameter.key}"
value="\${value}" />
                    </c:forEach>
                </c:forEach>
                <input type="hidden" name="partNumber"
value="\${WCPParam.partNumber}" />
                <input type="submit" name="review" value="Write a
review" />
            </fieldset>
        </form>
    </c:otherwise>
</c:choose>
<div class="sort_by_control">
    Sort by:
    <c:choose>
        <c:when test="\${reviews.sortType == 'rating' ||
reviews.sortType == 'default'}">
            <wcf:url var="ProductReviewListSortURL"
value="mProductReviewsView">
                <wcf:param name="langId" value="\${langId}" />
                <wcf:param name="storeId"
value="\${WCPParam.storeId}" />
                <wcf:param name="catalogId"
value="\${WCPParam.catalogId}" />

```

```

                                <wcf:param name="productId"
value="{WCParam.productId}" />
                                <wcf:param name="pgGrp"
value="{WCParam.pgGrp}" />
                                <wcf:param name="onPage"
value="{WCParam.onPage}" />
                                <wcf:param name="sortOptions"
value="submissionTime|desc" />
                                </wcf:url>
                                <span class="current_sort">Highest rating</span>
<a href="{c:out value="{ProductReviewListSortURL}"/>">Newest post</a>
                                </c:when>
                                <c:otherwise>
                                <wcf:url var="ProductReviewListSortURL"
value="mProductReviewsView">
                                <wcf:param name="langId" value="{langId}" />
                                <wcf:param name="storeId"
value="{WCParam.storeId}" />
                                <wcf:param name="catalogId"
value="{WCParam.catalogId}" />
                                <wcf:param name="productId"
value="{WCParam.productId}" />
                                <wcf:param name="pgGrp"
value="{WCParam.pgGrp}" />
                                <wcf:param name="onPage"
value="{WCParam.onPage}" />
                                <wcf:param name="sortOptions"
value="rating|desc" />
                                </wcf:url>
                                <a href="{c:out
value="{ProductReviewListSortURL}"/>">Highest
rating</a>&nbsp;&nbsp;&nbsp;<span class="current_sort">Newest post</span>
                                </c:otherwise>
                                </c:choose>
                                </div>

                                <ol class="list_reviews">
                                <c:forEach var="reviewDet"
items="{reviews.reviewDetailsMap}" varStatus="status">
                                <wcf:url var="readReviewLink" value="mReviewDetails">
                                <c:forEach var="parameter" items="{WCParamValues}">
                                <c:forEach var="value" items="{parameter.value}">
                                <wcf:param name="{parameter.key}"
value="{value}" />
                                </c:forEach>

```



```

        </c:forEach>
        <wcf:param name="reviewId"
value="\${reviewDet.value.reviewId}"/>
        </wcf:url>
        <li>
            <div class="container">
                <a href="#" class="user_image">images/avatar.jpg" width="34" height="34"
alt="\<out value="\${product.description.name}" escapeXml="false"/>"
/></a>

                <ul>
                    <li><c:import
url="\${jspStoreDir}mobile/Snippets/ReusableObjects/StarRating.jsp">
                        <c:param name="lowerBound"
value="\${reviews.lowerBound}" />
                        <c:param name="upperBound"
value="\${reviews.upperBound}" />
                        <c:param name="avgOverallRating"
value="\${reviewDet.value.rating}" />
                    </c:import></li>
                    <li><h4><c:out
value="\${reviewDet.value.title}" escapeXml="false"/></h4></li>
                    <li>by: <c:out
value="\${reviewDet.value.userId}" escapeXml="false"/> on
<fmt:formatDate pattern="M/d/yy h:mm"
value="\${reviewDet.value.submissionTimeAsDate}" /></li>
                    <li><span class="bullet">&#187; </span><a
href="\${readReviewLink}">Read review</a></li>
                </ul>
                <div class="clear_float"></div>
            </div>
        </li>
    </c:forEach>
</ol>

    <div class="paging_control">
        <wcf:url var="ProductReviewListPrevURL"
value="mProductReviewsView">
            <wcf:param name="langId" value="\${langId}" />
            <wcf:param name="storeId"
value="\${WParam.storeId}" />
            <wcf:param name="catalogId"
value="\${WParam.catalogId}" />
            <wcf:param name="productId"
value="\${WParam.productId}" />

```

```

        <wcf:param name="pgGrp" value="{WCPParam.pgGrp}" />
        <wcf:param name="onPage" value="{reviews.pageNum -
1}" />
        <wcf:param name="sortOptions"
value="{WCPParam.sortOptions}" />
    </wcf:url>
    <wcf:url var="ProductReviewListNextURL"
value="mProductReviewsView">
        <wcf:param name="langId" value="{langId}" />
        <wcf:param name="storeId"
value="{WCPParam.storeId}" />
        <wcf:param name="catalogId"
value="{WCPParam.catalogId}" />
        <wcf:param name="productId"
value="{WCPParam.productId}" />
        <wcf:param name="pgGrp" value="{WCPParam.pgGrp}" />
        <wcf:param name="onPage" value="{reviews.pageNum +
1}" />
        <wcf:param name="sortOptions"
value="{WCPParam.sortOptions}" />
    </wcf:url>
    <div class="page_number">Page <c:out
value="{reviews.pageNum}" escapeXml="false"/></c:out
value="{reviews.noOfTotalPages}" escapeXml="false"/></div>
    <c:if test="{reviews.noOfTotalPages > 1}">
        <c:if test="{reviews.pageNum > 1}">
            <span class="bullet">&#171; </span>
            <a href="{c:out
value="{ProductReviewListPrevURL}"/>" title="Previous Page">Prev</a>
            </c:if>
            <c:if test="{reviews.pageNum <
reviews.noOfTotalPages}">
                &nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<a href="{c:out
value="{ProductReviewListNextURL}"/>" title="Next Page">Next</a><span
class="bullet"> &#187;</span>
            </c:if>
        </c:if>
    </div>
</c:otherwise>
</c:choose>
</div>

<%out.flush();%>
<%@ include file="../../include/FooterDisplay.jspf" %>
</div>

```

```
</body>
</html>
```

B.4 PostReview sample code

Example B-4 shows the content of the PostReview.jsp sample code.

Example B-4 Sample code for PostReview.jsp

```
<%--
=====
Licensed Materials - Property of IBM

WebSphere Commerce

(C) Copyright IBM Corp. 2008, 2009 All Rights Reserved.

US Government Users Restricted Rights - Use, duplication or
disclosure restricted by GSA ADP Schedule Contract with
IBM Corp.
=====
--%>

<%--
*****
* This JSP takes the review inputs and submits to PostReviewCmd.
*****
--%>

<!-- BEGIN PostReview.jsp -->

<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="http://commerce.ibm.com/base" prefix="wcbase" %>
<%@ taglib uri="http://commerce.ibm.com/foundation" prefix="wcf" %>

<%@ include file="../../../include/parameters.jspf" %>
<%@ include file="../../../include/JSTLEnvironmentSetup.jspf" %>
<%@ include file="../../../include/ErrorMessageSetup.jspf" %>
```

```

<c:set var="starImagePath" value="/soccom/ibm/social/images/" />
<c:set var="productId" value="${WCPParam.productId}" />

<c:set var="pgGrp" value="${WCPParam.pgGrp}" />

<wbase:useBean id="catalogEntry"
classname="com.ibm.commerce.catalog.beans.CatalogEntryDataBean" />

<c:choose>
  <c:when test="${catalogEntry.bundle == true}">
    <wbase:useBean id="product"
classname="com.ibm.commerce.catalog.beans.BundleDataBean"
scope="request" />
  </c:when>
  <c:when test="${catalogEntry.package == true}">
    <wbase:useBean id="product"
classname="com.ibm.commerce.catalog.beans.PackageDataBean"
scope="request" />
  </c:when>
  <c:when test="${catalogEntry.item == true}">
    <wbase:useBean id="product"
classname="com.ibm.commerce.catalog.beans.ItemDataBean" scope="request"
/>
  </c:when>
  <c:when test="${catalogEntry.product == true}">
    <wbase:useBean id="product"
classname="com.ibm.commerce.catalog.beans.ProductDataBean"
scope="request" />
  </c:when>
</c:choose>
<c:set property="partNumber" value="${catalogEntry.partNumber}"
target="${WCPParam}" />

<!DOCTYPE html PUBLIC "-//WAPFORUM//DTD XHTML Mobile 1.2//EN"
"http://www.openmobilealliance.org/tech/DTD/xhtml-mobile12.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" lang="${shortLocale}"
xml:lang="${shortLocale}">
  <head>
    <title><fmt:message key="PRODUCT_TITLE" bundle="${storeText}" />:
<c:out value="${product.description.name}" escapeXml="false"/></title>
    <meta http-equiv="content-type" content="application/xhtml+xml"
/>
    <meta http-equiv="cache-control" content="max-age=300" />

```

```

        <meta name="viewport" content="width=device-width,
initial-scale=1.0, user-scalable=no" />
        <link rel="stylesheet" href="{cssPath}"/>
    </head>

    <body>
        <div id="wrapper">

            <%@ include file="../../include/HeaderDisplay.jspf" %>
            <%@ include
file="../../include/BreadCrumbTrailDisplay.jspf"%>

            <%out.flush();%>

            <div id="write_review" class="content_box">
                <div class="heading_container">
                    <h2>Write a Review</h2>
                    <div class="clear_float"></div>

                </div>
                <a href="javascript:void(0);"><c:out
value="{product.description.name}" escapeXml="false"/></a>
                <p><span class="field_required_symbol">*</span>Indicates
required fields</p>
                <c:choose>
                    <c:when test="{!empty storeError.key && storeError.key
== '_ERR_USER_NOT_LOGGED_IN'}">
                        <wcf:url var="logOnURL" value="MobileLogonForm">
                            <wcf:param name="catalogId"
value="{WParam.catalogId}" />
                            <wcf:param name="storeId"
value="{WParam.storeId}" />
                            <wcf:param name="langId" value="{WParam.langId}"
/>
                        </wcf:url>
                        <wcf:url var="registerURL"
value="MobileUserRegistrationAddForm">
                            <wcf:param name="catalogId"
value="{WParam.catalogId}" />
                            <wcf:param name="storeId"
value="{WParam.storeId}" />
                            <wcf:param name="langId" value="{WParam.langId}"
/>
                            <wcf:param name="register_button" value="Register"
/>

```

```

        </wcf:url>
        Review could not be created. <a
href="{fn:escapeXml(registerURL)}">Register</a> to create a new
review or <a href="{fn:escapeXml(logOnURL)}" title="{fmt:message
key="SIGN_IN" bundle="{storeText}" />"><fmt:message key="SIGN_IN"
bundle="{storeText}" /></a> in if you are already a member.
    </c:when>
    <c:otherwise>
    <c:choose>
        <c:when test="{!empty errorMessage}">
            <p class="error"><c:out value="{errorMessage}"
/></p>

        </c:when>
        <c:otherwise>
            <c:if test="{!empty storeError.key}">
                <p class="error"><c:out
value="{storeError.key}" /></p>
            </c:if>
        </c:otherwise>
    </c:choose>
    <form method="post" action="PostReview">
    <fieldset>
        <p><span class="field_required_symbol">*</span>Please
select your rating for this product</p>
        <p class="rating">
            <input type="radio" name="rating" value="1">
            
            </input><br>
            <input type="radio" name="rating" value="2">
            
            </input><br>
            <input type="radio" name="rating" value="3">
            
        </input><br>
        <input type="radio" name="rating" value="4">
        
        </input><br>
        <input type="radio" name="rating" value="5">
        
        </input>
    </p>

    <div class="input_container">
        <div><label for="review_title"><span
class="field_required_symbol">*</span>Enter a title</label></div>
        <input type="text" id="review_title" name="title"
class="coloured_input" />
    </div>

    <div class="textarea_container">
        <div><label for="review_comments"><span
class="field_required_symbol">*</span>Comments</label></div>
        <textarea id="review_comments" name="body"
class="coloured_input" rows="8"></textarea>
    </div>
    <input type="hidden" name="partNumber"
value="{WCParam.partNumber}" />
    <c:forEach var="parameter" items="{WCParamValues}">
        <c:forEach var="value" items="{parameter.value}">
            <input type="hidden" name="{parameter.key}"
value="{value}" />

```

```

        </c:forEach>
    </c:forEach>
    <input type="hidden" name="URL"
value="mPostReviewsView" />
    <input type="hidden" name="viewTaskName"
value="mPostReviewsView" />
    <input type="hidden" name="errorViewName"
value="mPostReviewsView" />
    <input type="hidden" name="onPage" value="1" />
    <input type="hidden" name="sortOptions"
value="submissionTime|desc" />
    <input type="submit" id="create_review"
name="create_review" value="Create" class="submit" /> <input
type="button" id="cancel_review" name="cancel_review" value="Cancel" />
    </fieldset>
</form>

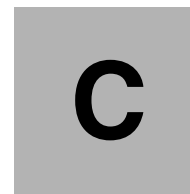
    <p>Content created on this site is subject to a delay in
viewing due to administrative reasons.</p>
    </c:otherwise>
</c:choose>

</div>

    <%out.flush();%>
    <%@ include file="../../include/FooterDisplay.jspf" %>
</div>
</body>
</html>

<!-- END PostReview.jsp -->

```



Additional material

This book refers to additional material that you can download from the Internet as described in this appendix.

Locating the Web material

The Web material that is associated with this book is available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser at:

<ftp://www.redbooks.ibm.com/redbooks/SG247787>

Alternatively, you can go to the IBM Redbooks Web site at:

ibm.com/redbooks

Select the **Additional materials** and open the directory that corresponds with the IBM Redbooks form number, SG247787.

Using the Web material

The additional Web material that accompanies this book includes the following files:

<i>File name</i>	<i>Description</i>
socom.zip	Compressed soccom code samples
DOMMediationModule-MB.zip	Compressed DOM code samples
Google.zip	Compressed store locator samples for Google
MapQuest.zip	Compressed MapQuest code samples

How to use the Web material

Create a subdirectory (folder) on your workstation, and decompress the contents of the Web material compressed files into this folder.

Related publications

We consider the publications that we list in this section particularly suitable for a more detailed discussion of the topics that we cover in this book.

IBM Redbooks publications

For information about ordering these publications, see “How to get Redbooks” on page 511. Note that some of the documents referenced here might be available in softcopy only.

- ▶ *WebSphere Commerce Best Practices in Web 2.0 Store*, SG24-7647
- ▶ *WebSphere Commerce High Availability and Performance Solutions*, SG24-7512
- ▶ *WebSphere Commerce Line-Of-Business Tooling Customization*, SG24-7619

Online resources

The following Web sites are also relevant as further information sources:

- ▶ Order flow process
<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.user.doc/concepts/cosoflow.htm>
- ▶ Get Order
<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.webservices.doc/refs/rwvgetorder.htm>
- ▶ SOI and BOD service modules
<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.developer.soa.doc/concepts/csdcompare.htm>
- ▶ Non-ATP inventory information model
http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.data.doc/concepts/cin_iminventoryasset3.htm

- ▶ Using the Management Center
<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.management-center.doc/tasks/ttfgeneral.htm>
- ▶ Using the DistributedMap and DistributedObjectCache interfaces for the dynamic cache
http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp?topic=/com.ibm.websphere.nd.doc/info/ae/ae/tdyn_distmap.html
- ▶ Object cache instance settings
http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/topic/com.ibm.websphere.nd.doc/info/ae/ae/udyn_cacheinstancescollection.html
- ▶ Class DistributedObjectCache
http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/topic/com.ibm.websphere.javadoc.doc/public_html/api/com/ibm/websphere/cache/DistributedObjectCache.html
- ▶ IBM Extended Cache Monitor for IBM WebSphere Application Server technology preview
http://www.ibm.com/developerworks/websphere/downloads/cache_monitor.html
- ▶ Data service layer
<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/topic/com.ibm.commerce.developer.soa.doc/concepts/csddsl.htm?resultof=%22%44%73%6c%22%20%22%64%73%6c%22%20>
- ▶ Business Object Document (BOD)
http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.base.doc/misc/Business_Object_Document_%28BOD%29.htm
- ▶ Get Request and the Show Response
<http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.webservices.doc/concepts/cwvget.htm>
- ▶ Dojo
<http://www.dojotoolkit.org/docs>
- ▶ Get Request and the Show Response
http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.web20storesolution.refapp.doc/concepts/csm_web20_intro.htm

- ▶ Web 2.0 store solution

http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.web20storesolution.refapp.doc/refs/rsm_web20_storepages.htm

- ▶ Enabling Distributed Order Management (DOM) integration

http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/index.jsp?topic=/com.ibm.commerce.dom-integration.doc/tasks/tsmbopisinstall_dup.htm

How to get Redbooks

You can search for, view, or download Redbooks, Redpapers, Technotes, draft publications and Additional materials, as well as order hardcopy Redbooks publications, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Index

A

- access control framework 394
- access control policy 250, 395
- Accessible rich Internet application (AHIA) 37
- accordion widget 34
- AcknowledgeInventoryRequirement 91, 117
- AcknowledgeOrder 91
- acpload 395–396
- acpload command 397
- acpload.log 252, 397
- Action 67
- action code 92
- action mappings 394
- Activity Builder 72
- activity templates 70
- add-on store archive 47
- addPaymentInstruction service 239
- address book 51
- Advanced B2B Direct starter store 47
- Ajax 40, 44
- Ajax checkout 40
- Ajax service 130
- AllSiteUsersViews action group 248
- analytics 6
- Android 179
- application and process integration 10
- application development 10
- application infrastructure 10
- ARTS 24
- Association for Retail Technology Standards (ARTS) 15, 22, 426
- ATP, *see* available to promise (ATP).
- attribute dictionary 46
- attribute filtering 63
- auctions 46
- available to promise (ATP) 30, 78–79, 81, 83, 86, 134
 - inventory management system 81
 - non-ATP inventory 81
 - non-ATP inventory management system 81
- avgOverallRating parameters 359

B

- B2B contracts 399
- B2C store 30
- back-end system integration 84
- basic composite store archive 38
- BazaarVoice 54, 58
- behavioral track 4
- Best deal calculation 64
- biConfig.xml 401
- billing address create view 237
- billing address selection view 237
- Blackberry 179
- BOPIS, *see* buy online, pick up in store
- Branch 68
- brick-and-mortar store 30–31
- BrowseDepartments.jspf 184
- browsing 7
- budget 2
- Business Activity Monitoring (BAM) 17
- business logic and views 38
- Business Object Document (BOD) 91
- business object schema 74
- business objects 74
- business-to-business starter stores 46
- buy online, pick up in store 19–21, 30–31, 83, 86, 134
- buy online, return in store 19

C

- CachedFooterDisplay.jsp 184
- CachedHeaderDisplay.jsp 184
- CachedItemDisplay.jsp 350
- CachedProductDisplay.jsp 348
- call center 78
- call center support 7
- campaigns 6
- CancelInventoryReservation 92
- Cascading Style Sheet (CSS) 37
- cascading style sheet (CSS) 390
- catalog browsing 86
- catalog entry 74
- catalog group 74
- catalog manager 46

- CatalogEntryDataBean 346
- CatalogSearchResultsDisplayView 184
- CategoriesDisplay 184
- category name 418
- category pages 32
- Change Flow pages 40
- ChangeOrder service 80, 89
- channel events 10
- Check Store Availability area 138
- checking inventory availability 78
- CheckInventory 92
- Checkout and Ship to Home/Office 51
- checkout pages 51
- collaboration 10, 13
- collaborative learning 13
- commerce 10
- Commerce Product Strategy 1
- common customer usage 19
- common1_1.css 391
- component client API 90
- ConsumerDirect 30
- ConsumerDirect starter store 40, 47
- ContentAreaESpot.jsp 184
- ContentPane widget 132
- control path 71
- ControllerCommand 377
- ControllerCommandImpl 377
- cookie contents 67
- Coremetrics 399, 402
- Coremetrics JSP tags 403, 405
- Coremetrics server 414
- Coremetrics Web Analytics 400
- cost benefit analysis (CBA) 425
- coupon wallet 36, 51
- cross-channel 1–2
- cross-channel integration solution 84
- cross-channel retailing solution 11
- cross-channel selling 11
- cross-channel solution 2
- CSR 22
- CSV input file 74
- current page 67
- Customer Care 46
- customer counter 72
- customer segments 71
- customer service representative (CSR) 8, 46
- customer-facing store 40

D

- data bean 324
- data beans 43
- Data load 74
- data load 74
- data service layer (DSL) 85
- database integration 10
- day and time 67
- DB2 75, 85
- DB2 Universal Database 99
- DecrementCache 92
- DEFBKDB6 100
- Delicious Web site 54
- delta load 74
- demand generation 6
- departure page view 418
- derby database 158
- Dialog Activity 68
- Digg 53
- dijit.Menu widget 39
- dijit.MenuItem instance 39
- display block 302
- distributed order management (DOM) 17, 19, 21, 30–31, 39, 83, 130–131, 134–135, 455
 - inventory management module 90
 - inventory system 90
- DistributedMap 85
- DistributedObjectCache 85
- Dojo dijit.Dialog 38
- Dojo framework 130
- Dojo parser 39
- Dojo Toolkit 38
- Dojo widgets 37–38
 - WCDialog 38
 - WCDropDownButton 39
 - WCMenu 39
- DOM, *see* distributed order management (DOM).

E

- ECAApplicationException 379
- Elite 30, 46–47
- Elite starter store 130, 135
- e-mail 6
- e-Marketing Spot 67–69, 73, 178, 181–185, 191–192, 199–201, 442–443, 452
- e-Marketing Spot report 399
- Enterprise Application Integration (EAI) 8
- enterprise architecture 11

Enterprise Architecture Frameworks 15
enterprise security 10
enterprise service bus 98
enterprise service bus (ESB) 19
entry page view 418
environmental concerns 2
Experiment 71
experiment path 71
External OMS simulator 128
external site referral 66

F

Facebook Web site 54
Fast Finder 33
FeaturedProductsESpot.jsp 184
filter catalog 63
financial crisis 2
findCurrentShoppingCart service 269
first-time shoppers 31
French (fr_FR) 45
fulfillment partners 2
fulfillment system 84

G

geoCodeLatitude parameter 208
geoCodeLongitude parameter 208
GeoCoder 208
geolocation API 212
German (de_DE) 45
getInventory_ExtOMSSim 116
GetInventoryAvailability 91, 93
GetInventoryAvailability service 90
GetInventoryAvailability services 83
GetOrder service 80, 89
Gift Center 46–47
gift registry 46
Global Positioning System (GPS) 208
Google Bookmarks 54
Google Map API 209
Google static map 215
GPSSupport.jspf 208
GPSSupport.jspf. 214
growth market slowdown 2
guest shopper 297

H

HttpClient 329

Hypertext Transfer Protocol (HTTP) 42, 50, 193,
324, 329, 376, 435

I

IBM Cloudscape 252
IBM Lotus Connections V2.5 58
IBM Rational Application Developer 75
IBM Rational Application Developer V7.5 75
IBM Retail Integration Framework 1, 11, 15, 24
IBM SOA Foundation Architecture 15
IBM WebSphere Application Server 75
IBM WebSphere Commerce V7 29, 69
IBM_UsableShippingInfo access profile 269
Import StarRating.jsp 359
impulse purchase 31
initial data load 74
in-store fulfillment component 20
in-store offers 6
in-store pick up service 20
inventory 74, 84
inventory availability 138
inventory cancellation 90
inventory query 90
inventory reservation 90
inventory system 6, 30
InventoryRequest 113
InventoryService 113
InventoryServices 113
invocation service 90
iPhone 179
Italian (it_IT) 45

J

J2EE Connector Architecture (JCA) 50, 193
Japanese (ja_JP) 45
Java compute node 117
Java EE 5 75
Java Standard Tag Library (JSTL) 41
JavaScript 44, 240
JavaScript API 151, 166
JavaScript API proxy 171
JavaServer Pages (JSP) 37, 154, 171, 181, 184,
195, 198, 200, 208–209, 215, 231–233, 236,
238–239, 241
JavaServer Pages (JSP) fragment 181
JavaServer Pages Standard Tag Library (JSTL)
208, 338
JCASMS-HTTP 50

JCASMS-WS 51
JSON API 331
JSON object 331
JSP tags 44

K

key applications 16
Key business processes 16
key channels 16
key information 16
key roles and business units 16
kiosks 78
Korean (ko_KR) 45

L

LDAP 376
live regions 37
local inventory 9
lowerBound 359
LTPA 376
LtpaToken2 cookie 375

M

macroeconomic conditions 2
Madisons 46
Madisons Mobile add-on store archive 47
Madisons Mobile starter store 51
Madisons mobile starter store 47–48, 178–179, 390, 399
Madisons starter store 38, 44, 83, 130, 132, 135, 179
Madisons Web 2.0 30
MadisonsMobile.sar 47
MadisonsMobile.sar, 47
Management Center 60, 62, 69, 195, 199
Manhattan Associates 30
Map API key 209
Map Application of iPhone 219
map service provider 209
mapping node 117
MapQuest 171
MapQuest Geocoding 166
MapQuest Geocoding API 171
marketing engine 71
marketing experimentation summary 399
mass marketing 5
massload tool 74

master data 11
master data management 24
mediation module 110
merchandise association 74
merchandising associations 51
message mediation module 98
message transformation module 90
messaging 10
mini shopping cart area 38
Mobile Catalog Browsing Flow 185
Mobile Checkout Flow 185
Mobile Commerce 178
mobile commerce 2
mobile device 47, 413
mobile device optimizations 179
mobile devices 2
Mobile Marketing 178
mobile pages 51
mobile phone 78
mobile shopper targeting 179
Mobile Shopping List 178
Mobile Store Locator 208
Mobile Store Locator Flow 185
mobile triggers 178
mobile Web page 181
Mobile Wish List 178
MobileApparelFeaturedProducts 182
MobileFurnitureFeaturedProducts 182
MobileHome.jsp 184
MobileHomePage 182
MobileHomePageFeaturedProducts 182
MobileKitchenwareFeaturedProducts 182
mobileOK Basic Tests 50
MobileTablewareFeaturedProducts 182
mOrderBillingAddressSelection 237, 248
mOrderBillingDetails 237
mOrderItemDisplay 237
mOrderPaymentDetails 237
mOrderShippingAddressSelection 241
mOrderShippingBillingConfirmationView 237
mOrderShippingBillingSummaryView 237
mOrderShippingDetails 241
mOrderShippingInstructions 241
mOrderShippingMethodSelection 241
Mozilla Firefox 54
MQ system tray icon 99
multichannel 2, 4
multichannel allocation 4
multichannel retailing 5

multichannel solutions 3
My Account pages 40
MySpace 54

N

new payment method 37
Nokia 179

O

OMS simulator 98
online behavior 67
online credit card information 37
online shopper 87
onPage property 357
Open Applications Group (OAG) 91
Oracle 85
order capture 78, 80
order confirmation page 303
order confirmation view 237
order integration 80
order lifecycle 18
order management 80
order management system 18, 78, 84
order status 181
order summary page 299
order summary view 237
OrderChangeServicePIAdd struts 235
OrderChangeServicePIAdd struts action 239
OrderChangeServiceShipInfoUpdate struts action 239
OrderShippingMethodSelection.jsp 278
outbound service messages 90
outgoing service messages 90
out-of-stock items 6
owner message format 90

P

page developers 39
page name 418
page size 36
page view 418
page view analytics 404
pagename 409
pageSize property 357
pageview tag 409
Parlay X compliant 51
Parlay X interface 193, 199

payment capture 37
payment details view 237
payment method page 297
payment_method_form_creditcard 235
payment_method_form_payinstore 235
payments 9
performExecute method 380
persistent session 65
personalization 6
Personalization ID 65
personalized list of stores 31
pervasive computing 10
pervasive connectivity 2
photo and video gallery 56
Pluck 58
Pluck Web site 56
Point-of-Sale (POS) 17, 22, 202, 428
Polish (pl_PL) 45
polite 37
populate method 328
portal 10
Portuguese (pt_BR) 45
PostReview.jsp 369, 375
PostReviewCmdImpl 375
post-transaction 7
Precision Marketing 65, 178
precision multichannel marketing 4
pre-store activities 6
price 74
ProcessInventoryRequirement 91, 117
ProcessOrder 91–92
ProcessOrder service 80, 89
product and category blogs 56
product attributes 63
product compare page 315
product Quick Info 32, 40
Product Quick View panel 32
product ratings and reviews 54
product thumbnail image 32, 40
product tracking 2
ProductCompareResultGridDisplay.jsp 353
progress indicators 33
Project Zero 58
promotion 202
promotional products 63
Promotions 60
promotions 6
promotions Summary 399
public bookmark 54

purchase decision 7

Q

Quick Checkout profile 51

R

RatingDisplay.jspf 345
ratings and reviews 54, 319
RatingsDisplay.jspf 347
real-time communication 13
Redbooks Web site 511
 Contact us xv
refresh area widget 131
registered shopper 297
Remote Transaction Interface (RTI) 26
Representational State Transfer (REST) 58
Request for Quote (RFQ) 46
ReserveInventory 91–92
reserving inventory 78
REST based API 58
retail analytics 11
reversing inventory reservation 78
review details 317
review list 315
ReviewDetailBean 334
ReviewsDataBean class 346
RIA, *see* rich Internet application (RIA).
rich Internet application (RIA) 30, 32–33, 37–38, 40, 47
Romanian (ro_RO) 45
rotating circle ball animation 33
Russian (ru_RU) 45

S

Sales Center 399
SampleDOMMediation 102
SampleDOMMediationFlow 102
SampleDOMMediationMessageSet 102
SampleExtOMS 107
screen reader support 37
screen readers 37
seamless shopping experience 5
search engine optimization (SEO) 41
searchHeader.jspf 184
SEO URLMapper 42
service client 89
service consumer 90

service-oriented architecture (SOA) 1, 44, 80, 421
service-oriented modelling and architecture (SOMA) 16, 426
services consumer 98
Services list 99
sessions 418
setRequestProperties 378
shipInstructionsError div 283
shipMethodError 271
shipping 51
shipping address display 305
shopper's address book 233
shopping cart pagination 36
Shopping Cart view 237
Shopping Cart, Wish List and Product Compare 34
Short Message Service (SMS) 6, 48, 50, 147, 178, 195, 451
ShowInventoryAvailability 91, 95
ShowMap.jspf 209
Simplified Chinese (zh_CN) 45
single inventory pool 3
site flow 181
 catalog subsystem 181
 main subsystem 181
 member subsystem 181
 order subsystem 181
SmartDataBeanImpl 324
smarter consumer 2
sMash 324
sMash adapters 58
SMS gateway 50–51
SOA Blueprint from ARTS 15
SOA Logical Model 15
SocApp 323
Socom 319
socom 323
Socom application 321
SocCore 323
Social Commerce 52–53, 58
social commerce 6
Social Commerce participation 67
social commerce world 2
social networking 52, 54
social profile 57
sortOptions property 357
Spanish (es_ES) 45
standard ASCII order 248
standard page flow for checkout 221
StarRating.jsp 337, 340

- Statistics 71
- Stock Locator 178
- stock locator 20, 31
- store administrators 132
- store ID 237
- store location 86
- Store Locator 178, 181, 188
- store locator 9, 31
- storefront 78
- storeOptions.jspf 184
- store-wide access keys 50
- struts configuration 393
- struts-config.xml 195
- struts-config-ext.xml 393
- stylesheet 118
- submitPaymentInfo 234
- suppliers 2
- supply chain 11
- SyncInventoryAvailability 95
- SyncOrder 95
- SyncOrder service 80, 89

T

- target 66
- target customers 67
- technology 2
- Traditional Chinese (zh_TW) 45
- transact 7
- TransferOrder 91–92
- Trigger 69

U

- unified customer-centric experience 2
- updateInventoryReservations 117
- updateInventoryReservationsResponse 117
- updateOrderShippingInfo service 239
- upperBound 359
- URLMapper.xml 42
- US English (en_US) 45

V

- validateParameters method 379
- ValidatePaymentMethodCmd task 297
- value proposition 1
- verb get 93
- ViewMap.jspf 219

W

- WAI-ARI 37
- WAI-ARIA 37
- wait trigger 68
- WBRK6_DEFAULT_BROKER 100
- WBRK6_DEFAULT_CONFIGURATION_MANAGE
R 100
- WBRK6_DEFAULT_QUEUE_MANAGER 100
- wc.render.declareContext 131
- wc.render.declareRefreshController 131
- wc.render.getContextById 131
- wc.render.getRefreshControllerById 131
- wc.render.updateContext 131
- wc.service.declare 44, 130
- wc.service.declareContext 45
- wc.service.declareRefreshController 45
- wc.service.getServiceById 131
- wc.service.invoke 130
- wc.widget.RangeSlider 131
- wc.widget.RefreshArea 131
- wc.widget.ScrollablePane 132
- WCDropDownButton widget 39
- wcf
 - declareRefreshController 45
 - declareRenderContext 45
 - declareService 44
- wcf tag library 41
- WCMenu widget 39
- Web 2.0 44
- Web 2.0 starter stores 130
- Web Accessibility Initiative (WAI) 37
- web activity diagram 66
- Web Analytics 399
- Web Content Accessibility Guidelines (WCAG) 37
- Web Services 14, 51
- WebSphere Commerce 11, 17, 21, 78, 130
- WebSphere Commerce Accelerator 40, 83
- WebSphere Commerce Accelerator tool 132
- WebSphere Commerce and BPM 17, 22
- WebSphere Commerce and Store 17
- WebSphere Commerce campaigns report 399
- WebSphere Commerce SOA service client 89
- WebSphere Enterprise Service Bus 24
- WebSphere Enterprise Service Bus (WESB) 126
- WebSphere ESB 90
- WebSphere Integration Developer (WID) 126
- WebSphere Message Broker 90, 100, 102
- WebSphere Message Broker runtime 99
- WebSphere Message Broker toolkit 99

WebSphere MQ 24, 99
WebSphere Process Server 22, 26
WebSphere Remote Server 14, 22, 25
Windows Mobile 179
windows mobile simulator 413
Wish List 181

X

XSL transformation 117



Redbooks

Building Multichannel Applications with WebSphere Commerce

(1.0" spine)
0.875" <-> 1.498"
460 <-> 788 pages



Building Multichannel Applications with WebSphere Commerce

WebSphere Commerce order and inventory management systems and DOM

This IBM Redbooks publication discusses the value proposition of cross-channel solutions and describes the IBM Retail Integration Framework Commerce Product Strategy solution and service-oriented architecture as an enabler.

MapQuest and Google Maps integration with WebSphere Commerce

This book describes the latest features and techniques of IBM WebSphere Commerce Version 7. In it, we present an overview of the WebSphere Commerce order and inventory management systems, the distributed order management integration framework, and a sample DOM integration scenario.

e-Marketing Spots and promotions for mobile users

We discuss the Web 2.0 Madisons starter store and present a hands-on experience that integrates MapQuest with the WebSphere Commerce V7 Store Locator feature. We discuss how to use the mobile features that are included in WebSphere Commerce V7 to define e-Marketing Spots and promotion for mobile users. In addition, we demonstrate how to use Google Maps with the Store Locator feature on a mobile device.

We include in this book an example about how to apply WebSphere Commerce features on a cross-channel solution as applied at the Easy Hogar y Construcción home improvement retail company in South America. The scenario explains how to scale from an SOA store to a cross-channel business model.

This book is designed for use by WebSphere Commerce developers, practitioners, and solution architects in various industries.

**INTERNATIONAL
TECHNICAL
SUPPORT
ORGANIZATION**

**BUILDING TECHNICAL
INFORMATION BASED ON
PRACTICAL EXPERIENCE**

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks